CHALMERS

Design, implementation and evaluation of RWTP-Gateway for FTP based data transfer

Master of Science Thesis in Networks and Distributed Systems

WAJAHAT TARIQ

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, March, 2010

Design, implementation and evaluation of RWTP-Gateway for FTP based data transfer

WAJAHAT TARIQ

Examiner: Ali Salehson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden March 2010

CHALMERS University of Technology

Deutsche Thomson OHG
Hannover Network Protocols Lab

Design, implementation and evaluation of RWTP-Gateway for FTP based data transfer

by

Wajahat Tariq

A thesis submitted in partial fulfillment for the
degree of Master of Science

in
Networks and Distributed Systems
Computer Science and Engineering

Supervisor **Dr. Eduard Siemens**
Examiner **Ali Salehson**

March 2010

"As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now that we have gigantic computers, programming has become a gigantic problem. As the power of available machines grew by a factor of more than a thousand, society's ambition to apply these new machines grew in proportion, and it was the poor programmer who found his job in this exploded field of tension between the ends and the means. The increased power of the hardware, together with the perhaps more dramatic increase in its reliability, made solutions feasible that the programmer had not dared to dream about a few years before. And now, a few years later, he had to dream about them and even worse, he had to transform such dreams into reality! It is no wonder that we found ourselves in a software crisis "

E. Dijkstra (The Humble Programmer, "ACM Turing Award Lectures: The First 25 Years", Addison-Wesley, 1987, pages 17-32)

Abstract

The trend of video processing is increasing towards a digital arena and making use of distributions over the internet. Hence, movies in the near future will be more digitial and highly detailed. The new technology that is being used today is called as the 4K technology, where 4K refers to the high resolution of the movie which is 4096x2160 pixels. This resolution gives a more detailed image than what can be seen on commonly available HD-TVs, plasmas, computer screens or digital projection systems. This high resolution implies that a huge amount of data would be processed when transferred over the Internet, which in turn requires considerably large data rate, around 2.5 Gbps, in the presence of delay and packet loss. The Internet reliable transport protocol TCP cannot handle such data rate due to its limitations when considering its congestion control mechanism. This is where a Reliable WAN (Wide Area Network) Transport Protocol (RWTP), developed by the company "Deutsche THOMSON OHG", plays its role. RWTP has been shown to have the capability to transfer data at a rate of 5 Gbps over WAN even in the presence of delay and packet loss. Though, it lacks the ability to perform collaborative tasks with most commonly available applications like File Transfer Protocol (FTP). In this thesis work a proxy mechanism is implemented to make RWTP perform these collaborative tasks with one of the most commonly used applications; i.e. FTP in this case. An implementation scenario is completed in C++ and tested on 1 Gbps Ethernet network with 1 Gbps switches, 10 Gbps network routers and 10 Gbps network emulator to emulate delay and packet losses. Using 4K digital technology with 5000 frames where each frame is 50 MBytes gives a total of 250 GBytes data. The result of thesis work show that this amount of data can be transferred from Hannover, Germany to Los Angeles, United States in 42 minutes using RWTP-Gateway with FTP. On the other hand, with a direct FTP connection using TCP it takes 37 hours, but only 28 hours with a proxy-based FTP connection using TCP. Hence, RWTP-Gateway with FTP gives a significant reduction in transmission time by providing high speed data transfer with a factor of 50 in comparison with a direct FTP connection and with a factor of 40 in comparison with proxy-based FTP connection using TCP.

Acknowledgements

First of all I would like to thank Eduard Siemens at “Deutsche THOMSON OHG” for offering me this thesis in the first place. I found it to be a very good thesis which was both challenging and interesting. I also thank Andreas Aust for providing help, comments and suggestions not only with the implementation but also with the thesis report as well. I would also like to thank Jens Brocke for describing RWTP in detail. Thanks to Ali Salehson at CHALMERS University of Technology for being my thesis examiner and providing valuable comments on the report. Finally, I would like to thank my parents for their encouragement and support.

Contents

Abstract **ii**

Acknowledgements **iii**

List of Figures **vi**

List of Tables **viii**

Abbreviations **ix**

1 Introduction **1**

2 Background **4**

2.1 OSI model 4

2.2 TCP/IP 6

2.2.1 Transmission Control Protocol 7

2.2.1.1 Connection establishment 7

2.2.1.2 Reliable data transfer 8

2.2.1.3 Connection termination 8

2.2.1.4 TCP Congestion control 9

2.2.1.5 TCP congestion control extensions 10

2.2.1.6 Random Early Detection (RED) 10

2.2.1.7 Proxy-based TCP extensions of congestion control 10

2.2.2 User Datagram Protocol 10

2.3 File Transfer Protocol 11

2.3.1 Active mode 12

2.3.2 Passive mode 13

2.4 WAN acceleration 14

2.4.1 RWTP 15

2.5 Proxy servers 15

2.5.1 Classical application proxies 16

3 Design **18**

3.1 Control channel block 19

3.1.1 client-side proxy authentication block 20

3.1.2 FTP authentication block 21

3.1.3 Client-side proxy routing block 22

3.1.4 FTP command reference block 23
3.1.5 Control connection setup sequence 25
3.1.6 FTP initial command sequence 30
3.2 Data channel block 32
3.2.1 Data channel connection sequence 33

4 Implementation 38
4.1 Modules 39
4.1.1 ClientSideProxy 40
4.1.2 ServerSideProxy 40
4.1.3 CTCPServer 41
4.1.4 STCPClient 41
4.1.5 ProxyAuthentication 42
4.1.6 ProxyStream 43
4.1.7 ProxyCmdRef 44
4.1.8 RWTPClient 44
4.1.9 RWTPServer 45
4.1.10 ProxyDataConnection 45
4.1.11 ProxyRouter 47
4.2 Flow chart: client-side proxy 47
4.3 Flow chart: server-side proxy 50

5 Performance Evaluation 52
5.1 Network Emulator 53
5.2 TCP window size adjustment 54
5.3 Measuring TCP bandwidth performance with Iperf 55
5.4 Testing FTP using a direct TCP connection 56
5.5 Testing an indirect connection with TCP based-proxy 57
5.6 Testing an indirect connection with RWTP-based proxy 58
5.7 Performance Evaluation 59

6 Summary and Future work 62

A Network performance tables 63

B CPU usage plots 66

Bibliography 72

List of Figures

2.1 PDU 4
2.2 OSI Layers 5
2.3 TCP : Connection establishment 8
2.4 TCP : Connection termination 9
2.5 UDP header 11
2.6 Active mode 12
2.7 FTP and NAT 13
2.8 FTP : Passive mode 14
2.9 Proxy 15
2.10 Classical proxy 16

3.1 Proxy design 18
3.2 Control Channel Blocks 19
3.3 client-side proxy authentication 21
3.4 FTP authentication sequence 22
3.5 client-side proxy routing process 22
3.6 Path determination 23
3.7 FTP command reference process 25
3.8 Control connection setup sequence 26
3.9 Overall control connection setup sequence 29
3.10 Basic FTP commands 30
3.11 Initial command sequence 31
3.12 PASV command and its responce 34
3.13 Data channel connection sequence 35
3.14 Gateway scenario 36
3.15 Gateway scenario with RWTP 37

4.1 Implementation model 39
4.2 TCP WAN connection 42
4.3 RWTP WAN connection 45
4.4 Flow chart: client-side proxy 49
4.5 Flow chart: server-side proxy 51

5.1 The network model used for performance evaluation 53
5.2 10 Gbps Network Emulator 54
5.3 TCP Bandwidth measurement scenario with Iperf 56
5.4 Testing Direct connection 57
5.5 Testing an indirect connection with TCP based proxy 58

5.6 Testing an indirect connection with RWTP based proxy 58
5.7 Performance analysis 59
5.8 TCP vs RWTP with 0.1 percent packet loss 60
5.9 TCP vs RWTP with 0.3 percent packet loss 61

B.1 CPU Usage at FTP client 66
B.2 CPU Usage at FTP server 67
B.3 CPU Usage at Client side proxy (TCP as WAN connection) 68
B.4 CPU Usage at Server side proxy (TCP as WAN connection) 69
B.5 CPU Usage at Client side proxy (RWTP as WAN connection) 70
B.6 CPU Usage at Server side proxy (RWTP as WAN connection) 71

List of Tables

3.1 TCP Over WAN Performance Tuning 36

5.1 Iperf TCP Performance 56
5.2 Performance analysis table for TCP-proxy, RWTP-proxy and direct connection 61

A.1 Performance analysis table for TCP Proxy, RWTP Proxy and direct connection without emulator 63
A.2 Performance analysis table for TCP Proxy, RWTP Proxy and direct connection with 0.1 percent packet loss 64
A.3 Performance analysis table for TCP Proxy, RWTP Proxy and direct connection with 0.3 percent packet loss 65

Abbreviations

AECN	**A**nti **ECN**
BDP	**B**andwidh **D**elay **P**roduct
ECN	**E**xplicit **C**ongestion **N**otification
FTP	**F**ile **T**ransfer **P**rotocol
Gbps	**G**iga **b**its **p**er **s**econd
GBps	**G**iga **B**ytes **p**er **s**econd
GHz	**G**iga **H**ertz
HTTP	**H**yper **T**ext **T**ransfer **P**rotocol
ISO	**I**nternational **S**tandards **O**rganization
LAN	**L**ocal **A**rea **N**etwork
LLC	**L**ogical **L**ink **C**ontrol
MAC	**M**edia **A**ccess **C**ontrol
Mbps	**M**ega **b**its **p**er **s**econd
MBps	**M**ega **B**ytes **p**er **s**econd
NIC	**N**etwork **I**nterface **C**ard
OSI	**O**pen **S**ystem **I**nterconnect
PEP	**P**erformance **E**nhancing **P**roxy
RED	**R**andom **E**arly **D**etection
RFC	**R**equest **F**or **C**omments
RTT	**R**ound **T**rip **T**ime
RWTP	**R**eliable **WAN** **T**ransport **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
UDT	**U**DP-based **D**ata **T**ransfer
WAN	**W**ide **A**rea **N**etwork

Chapter 1

Introduction

As of June 2009, there are 1.66 billion Internet users according to Internet World Stats and the number is increasing day by day [1]. Also, bandwidth demanding applications with reliable data transfer are gaining popularity.

The most commonly used protocol for reliable data transfer over the Internet is TCP. It allows applications to communicate reliably over the unreliable IP packet-switching network. It does so while dynamically increasing its sending window size which represents the amount of data that can be sent without having to wait for their acknowledgement [2]. TCP suffers limitations in transfering over large distances because the mechanisms implemented for reliable data transfer and congestion control will be effected by long delays and packet losses.

Getting good TCP performance with high latency and high bandwidth needs tuning of the software implementation. A properly tuned TCP can achieve up to 100 Mbps but beyond that in a high latency environment its performance starts to degrade. The reason for this degradation in performance is TCP congestion control algorithm controlling the sending window size [3].

When considered with standard and even enhanced versions of TCP there is a huge gap between the available bandwidth and the bandwidth being utilized i.e. throughput.

TCP performance will also be degraded largely due to packet losses. Due to being reliable, TCP will ensure to receive acknowledgement otherwise it retransmits that part of data after time out. This retransmission process to recover from errors and losses causes the effective bandwidth to be dropped below the available bandwidth between the end devices.

Latency is an issue which is related to the distance and the speed of signal propogation since, the speed of signal propagation in vacuum is constant "299,792,458 meters/second" and in fibre is roughly 66 percent of speed of signal propogation in vacuum which is 200,000,000 meters/second [4]. Latency is directly proportional to the distance between two end points of communicating devices. Hence, the larger the distance, the more will be the delay. The distance for example between Los Angeles, United States and Hannover, Germany is 10,000 kilometers, the latency between them over transcontinental links becomes:

$$\text{latency} = \text{Distance} \ / \ \text{Speed} = 10{,}000/200{,}000 = 50 \text{ milli seconds}$$

The sending window size is always limited by the so called Bandwidth Delay Product (BDP). With a window size of 64 KB and 100 milli seconds delay the highest possible rate is around 5.6 Mbps. Further, the trend in professional movie production is moving towards a fully digital, file based and distributed workflow. High-quality film material for digital cinema capable of 4K resolution (4096x2160) requires approximately 2.5 Gbps for high speed transmisisons [5]. To transmit such amount of data over larger distances having a Round-Trip Time (RTT) of 100 milli seconds, a high speed reliable transport protocol is required.

Such a reliable transport protocol has been developed by the company "Deutsche THOMSON OHG" and is named Reliable Wide Area Network Transport Protocol (RWTP). This protocol has been shown to have the capability to transfer data at the rate of 5 Gbps in a single stream via transcontinental links with high delay even in the presence of packet losses.

An implementation of this protocol with a file transfer application is currently available for high-end Linux-based computer systems. Unfortunately, legacy equipment like Windows-based or Mac OS-based systems currently cannot benefit from RWTP's WAN performance. One possibililty to perform bulk data transfer over WAN with FTP in the presence of delay and packet losses is to use a proxy mechanism and enable it to adopt a high data transfer capability. Hence in order to investigate the advantages of RWTP, an RWTP-based gateway with FTP-based data transfer is designed, implemented and evaluated.

The implementation of proxy application for FTP-based data transfer is carried out in C++ using Linux and tested on a system with processor 2.4 GHz with 8 cores. The scenario is implemented on a 1 Gbps Ethernet network consisting of 1 Gbps switches and 10 Gbps network routers. In addition to this a 10 Gbit network emulator is used for performance analysis in the presence of delay and packet loss.

This report contains only minimal description about the proprietary protocol (RWTP) and its mechanisms. The full report is delivered to the company "Deutsche THOMSON

OHG", since they required some restrictions on the published content of the report. The remainder of this report is structured as follows. In chapter 2, background information related to the thesis is given. The background information includes some basic networking concpets leading towards TCP, UDP and the transport protocol RWTP. FTP and proxy mechanism is also explained to provide a basic foundation before moving towards the design phase which is explained in chapter 3. Design phase will describe the two core application blocks, control block and data block which will be further explained in detail along with functions and methods. Chapter 4 starts with an overview of the implementation of all the modules and flow chart is given at the end. Performance analysis in different scenarios and the corresponding results are discussed in chatper 5. Summary and future work is described in chapter 6. Appendix A consists of network performance tables and appendix B at the end contains the CPU and network performance plots.

Chapter 2

Background

Whenever two computers need to communicate with each other, they have to be connected and the flow of information from one computer to another has to follow some rule in a well-defined format or syntax. This syntax or the set of rules is known as a protocol [6]. Over the years many protocols are developed but for the two systems to communicate they should adopt the same set of protocol. These protocols include mechanisms for the devices to identify and make connections with each other so that they will be able to communicate data.

Communication between two systems is all about transfering data from one system to another in the form of Protocol Data Unit (PDU) as shown in figure 2.1. Each PDU has some piece of control information added as header so that it could travel from one system to another [7]. The same protocol on each system will use the header information for its work.

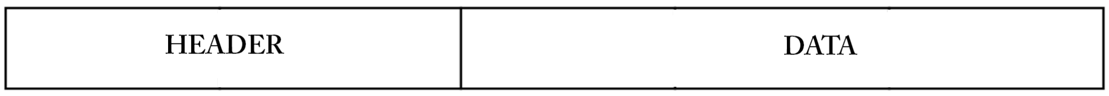


Figure 2.1: PDU

2.1 OSI model

OSI model has been an essential element of computer network design since its confirmation in year 1984 [8]. It is an abstract model which shows how network devices and protocols should work together and communicate with each other. It is a technology standard maintained by the International Standards Organization (ISO). In order to making systems communicate efficiently over the network, the entire communication task is divided into smaller functions in different levels which we refer to as layers. OSI model conceptually divides communication functions into seven layers where the lower layers deal with the electrical signals, chunks of binay data and perform routing across networks. On the other hand, higher layers deal with the requests and their responses, representation of data and all functions related to the user applications. It was originally conceived as a standard architecture for building network systems and indeed, many poplular network technologies today reflect the layered OSI model. The seven layers of OSI model are:

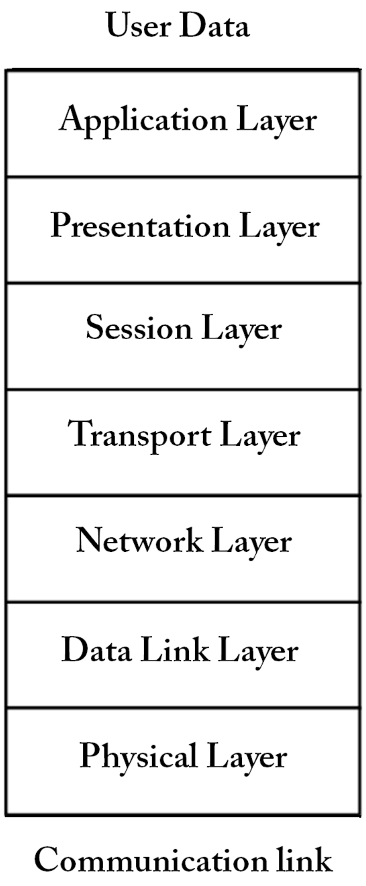


Figure 2.2: OSI Layers

The physical layer is responsible for transmitting raw bit stream over physical link. Also, it is responsible for converting bit stream into electrical signals by following suitable signalling and coding schemes.

It is designed to ensure reliable data transmission over a noisy physical channel [9]. This is usually done by adding a trailer for error control of the frame. It may also control the access to the physical medium between many devices.

The network layer is responsible for addressing and forwarding the packets. It has the task to find the ultimate destination for the packets and also performs the routing for so that packets could reach the correct destination.

The transport layer performs segmentation. If the received data from the upper layer is larger than the acceptable packet size, it divides the data into smaller chunks for transmission. It may perform retransmissions if the data is not received correctly at the destination. It is also responsible for sending the data at rate acceptable to the receiver i.e. it handles flow control otherwise it might overwhelm the receiver. Transport layer provides end-to-end transfer of data with optional types of control.

The session layer establishes, maintains and terminates a communication sessions which refers to the logical connection between two devices. It also recognizes the names so that only permissible parties participate in the sessions. These sessions enables systems on the network to share information.

The presentation layer is respnsible for providing data in a user-specific format which different systems use to store and manage their data. It is also responsible for encryption. In this way, the application has nothing to worry about the conversion or decryption of data.

The application layer is the highest layer of OSI model and its purpose is to serve as a window between two correspondent processes so that they may be able to exchange information in an open enviroment. Application layer is the interface between user application programs and the communication system so that the user programs will be able to access services that are available over the Internet.

2.2 TCP/IP

TCP/IP is a reference model and the protocol architecture that is developed for the Internet. It is a set of the most important protocols in the world. It is named for its two important protocols TCP and IP. TCP/IP provides communication services for the user applications across any underlying physical network. Two different transport layer protocols, TCP and UDP are discussed in this section.

2.2.1 Transmission Control Protocol

This protocol is layer 4 protocol which makes use of the services provided by the underlying Internet protocol IP (layer 3). TCP handles the data received from application layer are segments of variable length information which will be delivered in IP datagrams. One of the most commonly used transport layer internet protocol is Transmission Control Protocol (TCP). It fits into a layered protocol architecture just above a basic Internet Protocol [10] which provides TCP a way to send and receive segments of variable length of information enclosed in internet datagram [11]. TCP is reliable, it has error recovery mechanism. It is full featured transport layer reliable and connection oriented protocol. This is why TCP helps us to developing client and server applications. It provides transport layer addressing mechanism which helps applications to make multiple connections using single IP address.

2.2.1.1 Connection establishment

Before any data transfer the TCP performs a three way handshake. Normally a client will initiate a TCP connection to a destination server. The following scenario is occurred when TCP connection is established.

- Server must be in waiting state to accept an incoming connection.
- The client side TCP initiate the process by sending a segment with a SYN flag set to 1 which informs the server its initial sequence number for the data.
- The server acknowledges the client with a segment containing its initial sequence as well containing its own initial sequence number the segment will have both SYN and ACK flag set to 1.
- The client then acknowledge the server with a segment with the flag ACK set to 1.

The number of segments required for this process is three, that is why this is known as three way handshake.

After this stage, connection is successfully established. Data is reliably transferred and after that, connection is terminated.

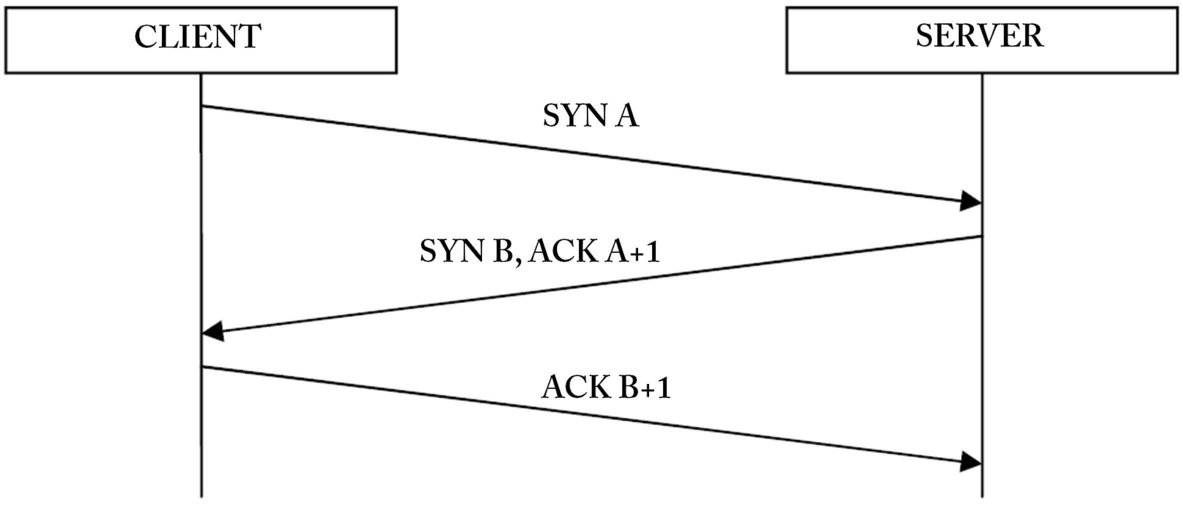


Figure 2.3: TCP : Connection establishment

2.2.1.2 Reliable data transfer

Data transfer in TCP is reliable because each segment has a sequence number and acknowledgement fields in the header. The sequence number is used by the sender to keep track of the data bytes that have been sent acknowledge or waiting to be sent. The receiver make use of the sequence number to order the received data and to acknowledge them to the ssender. If segments are not acknowledged, the sender may retransmit. It has a sliding window mechanism in which unacknowledged transmissions are detected and are retransmitted. TCP also uses the checksum to verify that the received data is error-free or not.

2.2.1.3 Connection termination

TCP connection termination is done by four segments as shown in the figure as shown in figure 2.4.

- When client wishes to close it sends a FIN segment set to 1, indicating that client has finished sending data and now wishes to close the connection.
- Server on the other end of the network sends ACK of the received FIN segment.
- Client will wait some time giving the sever the possibility to terminate its data sending. When the server has no more data to send it will include a FIN flag set to 1 in the last segment in which the client will act in the same way.

- The TCP on the client system sends an ACK back to it.

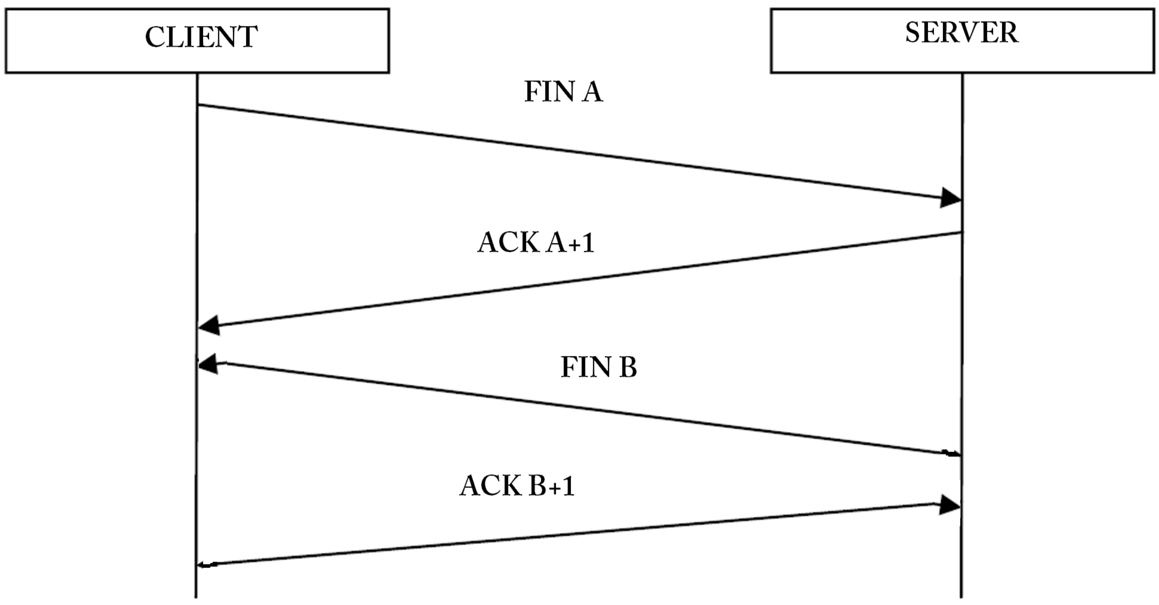


FIGURE 2.4: TCP : Connection termination

2.2.1.4 TCP Congestion control

In normal cases when TCP performing data transfer it does also congestion control in order to avoid the overwelming network by excessive packets. TCP starts after some threshold with incrementing its sending window size by one segment after each RTT (RTT). When it detects congestion due to missing ACKs from the receiver it decreases the window size by one half [12]. Considering a high bandwidth connection with high latency with 10,000 segments and since TCP reduces the window by one half in case of congestions therefore the window size now after congestion becomes 5000 segments. TCP then starts to increase one packet each RTT, if the RTT is 100 milli seconds then the time required to come back to 10,000 segments would be 500 seconds [13].

TCP congestion control mechanism is combined with error recovery mechanism. The arrival of an acknowledgement to the sender identifies that there is no congestion on the network and causes the window size to be increased by one segment. In case of normal congestion TCP will decrease its window by one half this is why its scheme is know as Additive Increase and Multiplicative decrease (AIMD).

2.2.1.5 TCP congestion control extensions

TCP congestion control can be performed by getting explicit or implicit notifications from routers or some other complex proxy-based methods.

2.2.1.6 Random Early Detection (RED)

Accross the network any router has the ability to drop packets only if its queue is full otherwise the packet has to stay a while in the queue and wait for its turn to be forwarded on its to destination. In case of congestion this queuing scheme becomes responsible for the loss of packets [14]. This packet loss due to queuing scheme has a negative effect on the fairness and throughput of TCP. In order to enhance the rate that the router drops packets in queue there are methods developed and generally called the active queue management one of these is RED algorithm which measures the mean queue length as control variable and compares it with two threshold queue length parameters. If mean queue length is less than the minimum queue threshold then packets are stored in the queue and later forwarded to the destination. On the other hand if mean queue length is more than the max queue threshold all arriving packets are dropped. If the mean queue length is between the upper and lower threshold then packets are dropped based on some probability. Hence, by using RED the fairness of TCP connections traversing the router is increased [15].

2.2.1.7 Proxy-based TCP extensions of congestion control

This is done by splitting a single TCP connection into two parts. The first part connects the sender to the proxy and the second part connects the receiver to the proxy. Also this proxy has the capability to act as sender and receiver in case it receives data from sending party and needed to send it to the recipient party. The advantage of using such a mechanism is that each TCP connection can be handled with a shorter response time. In case of congestion control mechanism is initiated this response time will be smaller than on bigger networks. This will give a much better performance of TCP. This type specific type of proxies is known as performance enhancing proxies (PEPs) [16].

2.2.2 User Datagram Protocol

User Datagram Protocol (UDP) is the simple transport protocol which provides similar addressing mechanisms like TCP but no reliability, is connectionless and data might get lost.

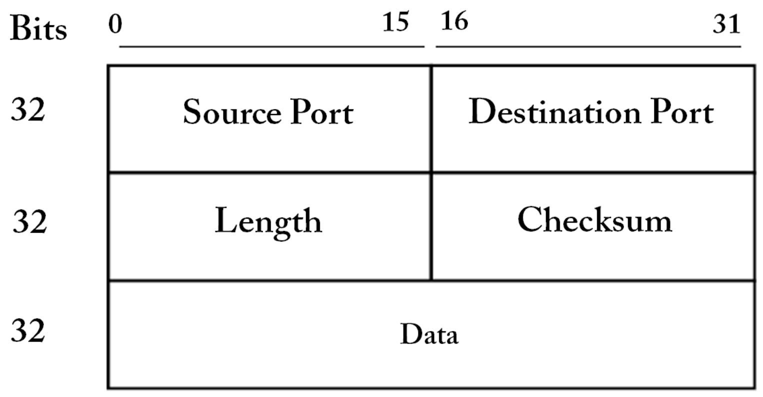


FIGURE 2.5: UDP header

UDP only provides multiplexing via port numbers and message integrity via checksum. Length field indicates the size of the segment header and the user data.

Therefore, UDP application must be willing to face some loss, errors, duplication or datagrams may also go out of order [17].

Even though it is less reliable, there are applications over the Internet making use of UDP. Some of them are Real Time Protocol (RTP) for real time multimdeia applications and TFTP (Trivial File Transfer Protocol) for transferring files.

2.3 File Transfer Protocol

File Transfer Protocol (FTP) is a protocol for transferring files over the internet. It is most commonly used to download a file from a server or upload to a server by making use of TCP connections [18].

FTP relies on a pair of TCP connections to perform its operation by establishing two different channels.

- Control channel
- Data Channel

Control channel is identified by TCP port number 21. All the commands are sent to FTP-server via this channel. Also, the responses of the commands are received via this control channel. For example, username and password to FTP-server will be sent on control channel “user abc”, “pass ****”. The responses are received on the same control channel “230 login successful” or “530 incorrect login”.

Data channel utilizes TCP port 20 for all data transfer between FTP client and FTP server. For example when the command for viewing directory “List -aL” is entered, the request is sent via control channel and the response of the command is sent on the data channel on TCP port 20.

Apart from FTP channels, there are two different modes in which FTP-client and FTP-server establishes a connection with each other.

- Active mode
- Passive mode

2.3.1 Active mode

In active mode, FTP-client connects to FTP-server by establishing a control connection on TCP port 21. FTP-client chooses a random port higher than 1023 to make connection with FTP-server on TCP port 21. When FTP-client wishes to make a data connection with FTP-server it sends a port command as shown in the figure 2.6, along with the IP address and port number to FTP-server to connect to. Also, FTP-client starts listening on that for the data channels.

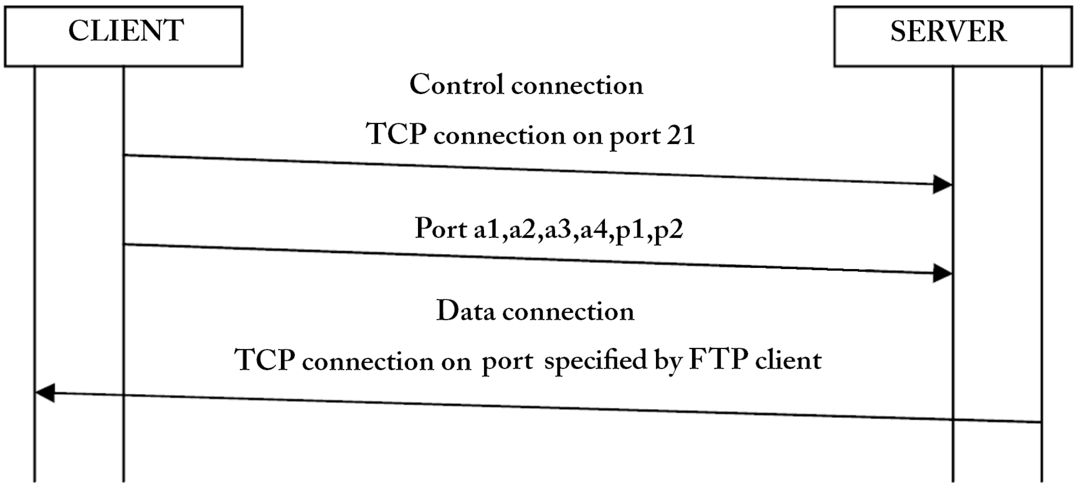

CLIENT
SERVER
Control connection
TCP connection on port 21
Port a1,a2,a3,a4,p1,p2
Data connection
TCP connection on port specified by FTP client

Figure 2.6: Active mode

FTP frequently fails when data has to pass through a firewall, because firewalls limit the use of number of ports to pass the unwanted traffic through them unless or until opened deliberately for specific connection in specific direction [19].

When FTP-client is behind the firewall, then active mode FTP will not work because the firewall might not know which of the clients behind it should receive the returned connection. The problem caused by FTP-client programs is by using PORT command to establish FTP data connections. In this case FTP-server has to make a connection back to FTP-client. But in the presense of restrictive firewalls which forbids every incoming connection will not allow a connection from FTP-server to FTP-client causing data transfer mechanism to fail [20].

Another problem with active mode FTP occurs when FTP-client program sends a PORT command to FTP-server from behind the Network Address Translator.

$$PORT \quad a1, a2, a3, a4, p1, p2 \tag{2.1}$$

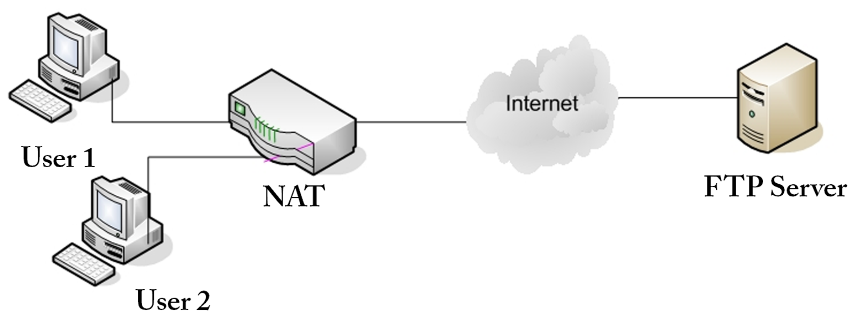


Figure 2.7: FTP and NAT

When this PORT command is sent to FTP-server and it tries to connect back to FTP-client, this request is dropped by the NAT router on the way to FTP-client. Because the port number is not existing in the translation table.

However, passive FTP works better in case of FTP clients protected by a firewall.

2.3.2 Passive mode

In passive mode, FTP-client connects to FTP-server to establish a data connection. First FTP-client makes a control connection with FTP-server on TCP port number 21.

It chooses a random port higher than 1023. When FTP-client wishes to transfer data, it sends a PASV command to FTP-server. As shown in the figure, FTP-server then responses with "227 Entering passive mode (a1,a2,a3,a4,p1,p2)". Where "a1.a2.a3.a4" is the IP address and "p1*256+p2" is the port number. After receiving the address from FTP-server, FTP-client makes a data connection to the port specified by the FTP-server as shown in figure 2.8.

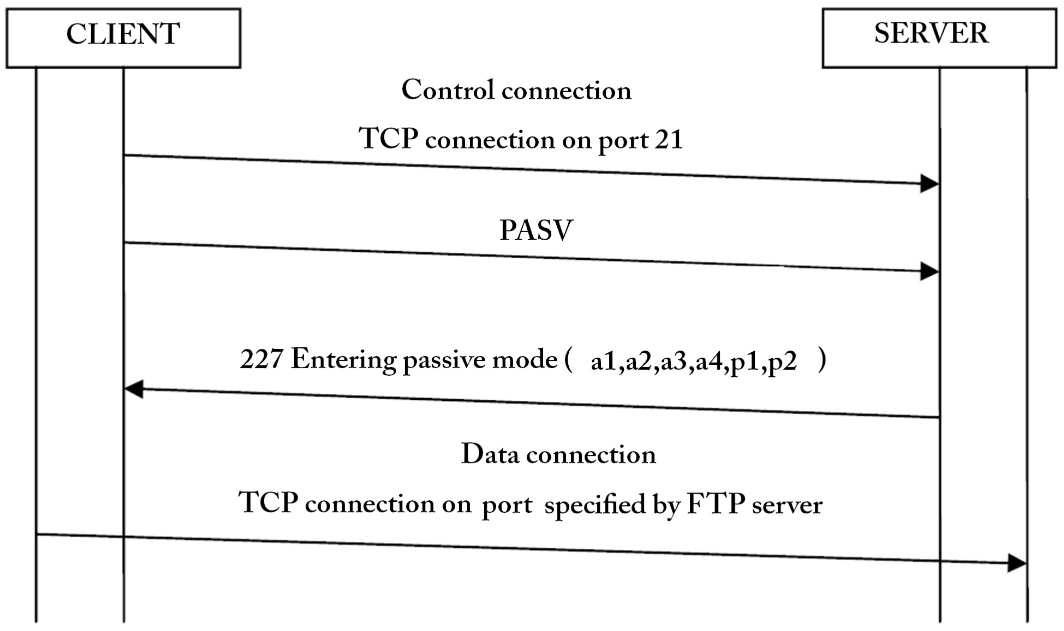

CLIENT
SERVER
Control connection
TCP connection on port 21
PASV
227 Entering passive mode (a1,a2,a3,a4,p1,p2)
Data connection
TCP connection on port specified by FTP server

FIGURE 2.8: FTP : Passive mode

Passive mode FTP implies that FTP-server does not make an attempt to establish data connection to FTP-client. Instead, FTP-client will initiate the data connection avoiding problems with firewall and NAT.

2.4 WAN acceleration

The applications that are used to work fine over the smaller network are now moving to global Internet and the major problem that these of applications are facing is longer responce time. Even working on the applications to improve their performance over WAN does not seem to be fruitful in some cases. Developers look for ways to increase the performance over WAN without having to make changes on the application itself [21].

Bandwidth range is extending and the capability of protocols in comparison to link speed is laging behind. For example, currently the fastest present standard is 10 Gbps

Ethernet. In novermber 2006, a group of IEEE agreed to a 100 Gbps Ethernet version. 40 Gbps networks are currently under development by IEEE [22].

One of the ways to perform a bulk data transfer over WAN in the presence of delay and packet losses is to use a proxy mechanism and enable it to adopt a high data transfer capability.

2.4.1 RWTP

RWTP (Reliable WAN Transport Protocol) is a transport protocol for high-speed data transmission over wide area networks developed by "Deutsche THOMSON OHG". It has the capability to support global data transfer of terabyte sized data sets [23]. RWTP can be considered as WAN protocol that is able to achieve a data rate of upto 2.6 Gbps over long distance connection that often exhibits high latency or packet loss. RWTP uses UDP to transfer data and extends it with properties such as reliability and flow control [24]. Thus, RWTP can be seen as a better version of TCP in terms of avoiding limitations of TCP over WAN.

One of the products utilizing RWTP is availabe and is used for data transport and storage access of motion pictures and television post production. It is well known as NetFlight adopted by Talon Data Systems [25].

2.5 Proxy servers

The term proxy means to act on the behalf of another. As shown in the figure 2.9, a proxy server is a system that lies in between client and server and forwards requests from client to servers and responses from servers to clients. When a request arrives at the proxy from client at first it analyzes the request, then it forwards the request to the server. Server sends the response back to the proxy. Again proxy analyzes the response and forwards it to the client [26]. By doing so, proxy applications help in providing anonymity to the client and server. Thus client can access the resources on the server and server can serve the requests for the client without knowing that each other are connected [27].

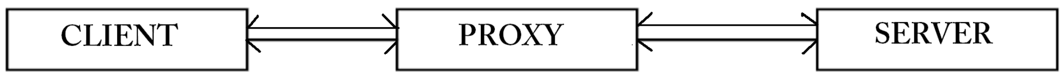


FIGURE 2.9: Proxy

2.5.1 Classical application proxies

A classical application proxy is a program that knows how to behave as both client and server. A classical proxy may modify or add a message on its way to client or server [28]. It implements the specific protocol being used by both client and server programs. In addition to that, the client must also have the knowledge to direct its request to the proxy and inform it about the final destination [29]. Essential qualities that a proxy server should have are as follows:

- It must be able to recognize clients and accept their connections.
- It must also understand the clients request and direct their requests to the final destination.
- It must be able to locate the final destination and make a connection with it.
- After establishing the connection with the client and the server, it must be able to relay requests and responses.

Hence, classical proxies not only provides security benefits among clients and servers but also provides the opportunity to make optimizations, logging and caching mechanisms to make the transfer be even more efficient.

Lets consider an example for classical application proxy. First of all, client must have the knowledge of classical existing proxy, so that it could direct its requests to the proxy instead of directing its requests directly to the server. Therefore, the client first makes a connection with proxy server. After making a successful connection with the proxy server it informs the proxy about the final destination address, to which the requests will be sent to.

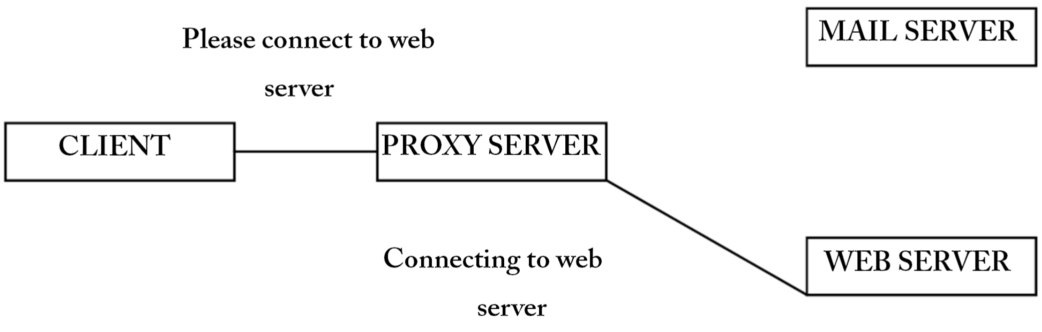


FIGURE 2.10: Classical proxy

Now the proxy server will locate the server and makes a connection with it.

As shown in the figure 2.10, client sends a request to proxy server and informs it that the final destination of the request is web server. Proxy server locates web server and makes a connection and forwards the request.

Chapter 3

Design

In this chapter, design of proxy application for FTP-based data transfer is discussed. The designing process is done under the consideration of the RFC 959 for File transfer Protocol [18] and RFC 1919 for classical and transparent proxies [29]. The design of the proxy application for FTP data transfer is performed in a way such that one proxy is placed near FTP-client and another is placed near FTP-server. The proxy near FTP-client is referred to as client-side proxy and similarly the proxy near FTP-server is referred to as server-side proxy. The scenario includes local networks of 1 Gbps connected by 10 Gbps routers via high-end Linux servers for client-side and server-side proxies as shown in figure 3.1.

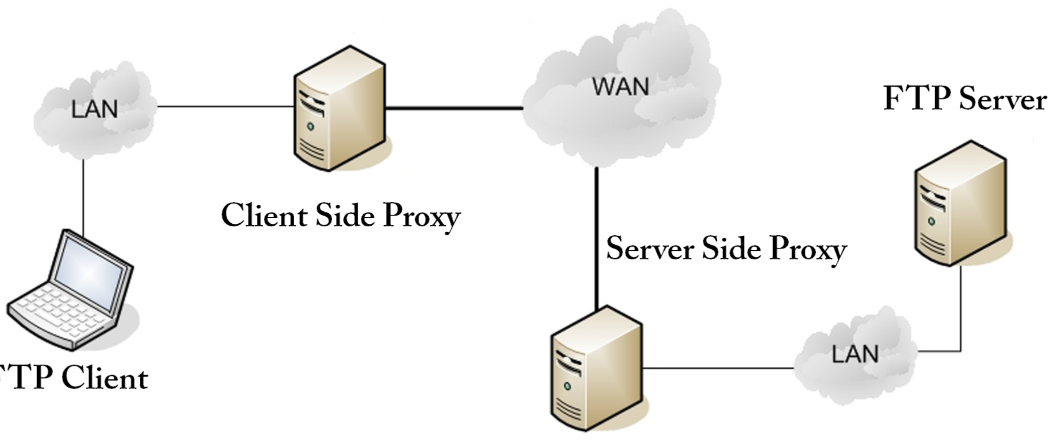


Figure 3.1: Proxy design

The proxy application for client-side and server-side proxy is divided into two main blocks.

- Control channel block
- Data channel block

3.1 Control channel block

Control channel block is further subdivided into four different blocks.

- Proxy authentication block
- Proxy routing block
- FTP authentication block
- Command reference block

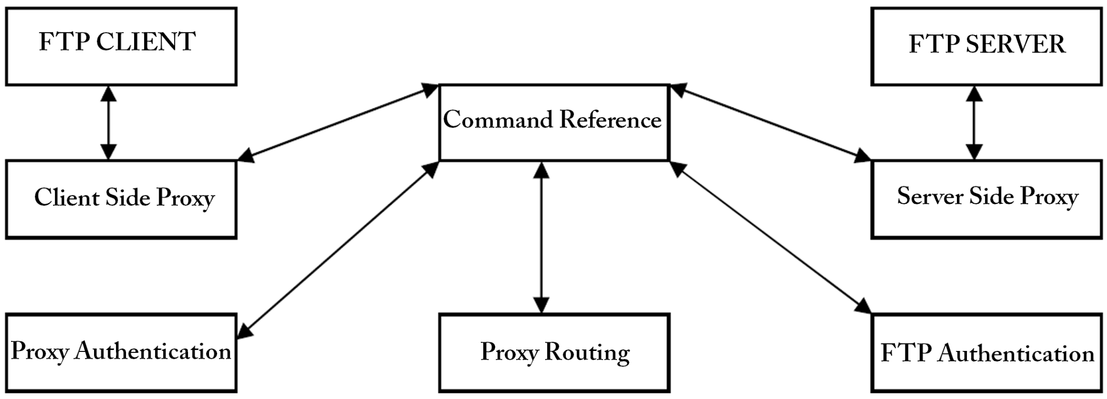


Figure 3.2: Control Channel Blocks

FTP-client is a client program which logs into FTP-server and sends requests over the network. In this case FTP-client should also have the capability to recognize the presence of proxy along the way to the FTP-server over the network. Hence, FTP-client first addresses the client-side proxy and then informs client-side proxy about the final destination FTP-server.

In our scenario, client-side proxy is the first communication point for FTP-client. Client-side proxy accepts a connection from FTP-client, also FTP-client informs client-side proxy about the final destination FTP-server. Client-side proxy also makes a decision about the server-side proxy based on the destination address.

Server-side proxy accepts incoming connection request from client-side proxy. Server-side proxy also receives final destination FTP-server address from client-side proxy. It makes a FTP connection with FTP-server.

As soon as connection of server-side proxy with FTP-server is established, FTP-server requests for authentication details from server-side proxy. FTP-server performs authentication and sends the login status back to FTP-client via server-side proxy.

3.1.1 client-side proxy authentication block

Proxy servers, apart from providing proxy services usually are also used to provide authentication [30]. FTP-client logs into client-side proxy after successful authentication. It sends its credentials after establishing a direct connection with client-side proxy first and after successful connection it send its credentials. First it sends username to client-side proxy as:

$$user \quad username \tag{3.1}$$

Client-side proxy after receiving username responds with "Please specify password" request to FTP-client and waits for password. FTP-client upon receiving password request from client-side proxy replies with password to client-side proxy as:

$$pass **** \tag{3.2}$$

After receiving username and password from FTP-client, client-side proxy performs authentication and sends back the status of login process back to FTP-client as shown in figure 3.3.

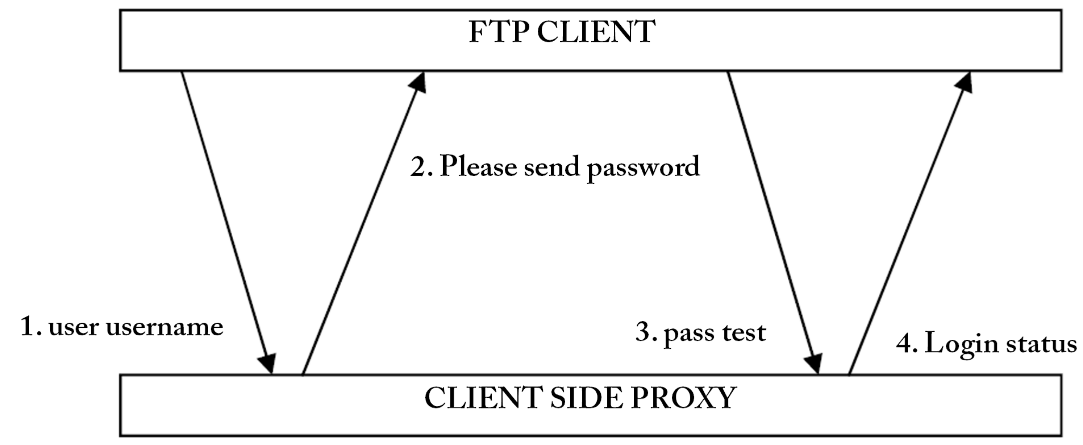


Figure 3.3: client-side proxy authentication

3.1.2 FTP authentication block

The authentication process on this architecture is performed after FTP-client connects with client-side proxy. Then FTP-client sends information about final destination to the server-side proxy in the form:

$$usename@ipaddress \tag{3.3}$$

Where username is the credential for authentication on FTP-server, “ipaddress” is the address of final destination FTP-server on which final destination FTP-server is listening on.

This information is passed on to server-side proxy which extracts the address and username of final destination FTP-server. It makes a connection with FTP-server and then after successful connection server-side proxy sends username to FTP-server.

After receiving username, FTP sever sends a “Please specify password” request to server-side proxy, which forwards it to client-side proxy and client-side proxy finally delivers it to FTP-client as shown in figure 3.4. FTP-client understands the request and sends back “pass ****” in response to password prompt request which reaches FTP-server via client-side and server-side proxy.

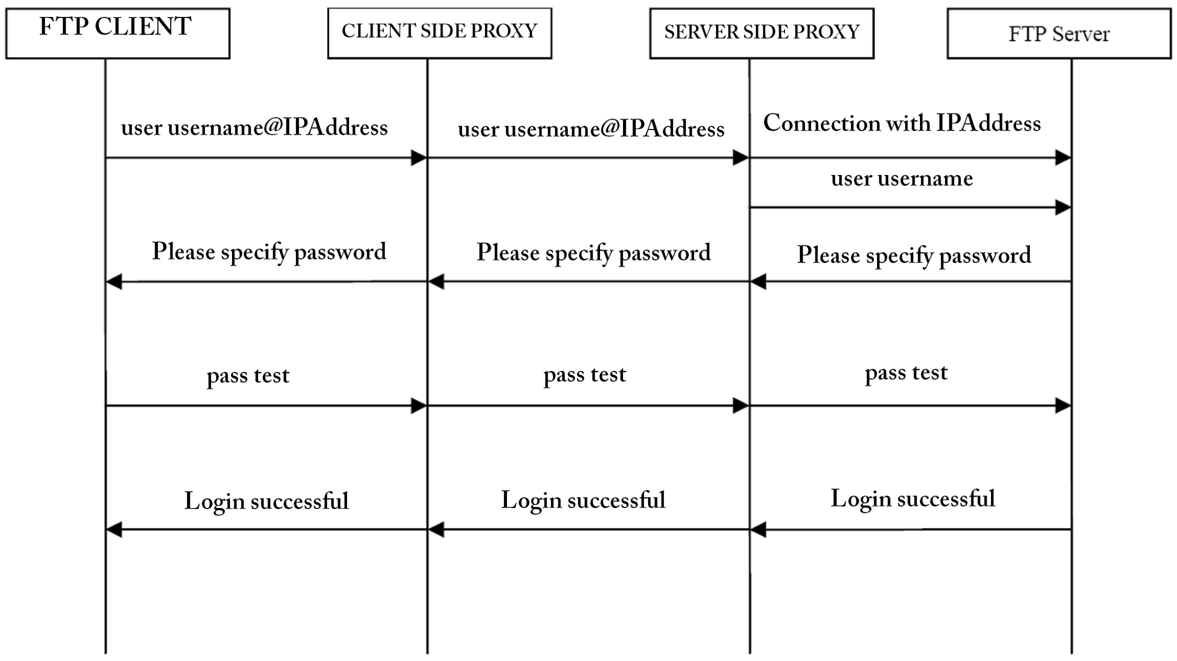


Figure 3.4: FTP authentication sequence

3.1.3 Client-side proxy routing block

Since there are a number of FTP-servers involved and also there are different server-side proxies for these FTP-servers therefore client-side proxy must make a decision about which server-side proxy to connect to.

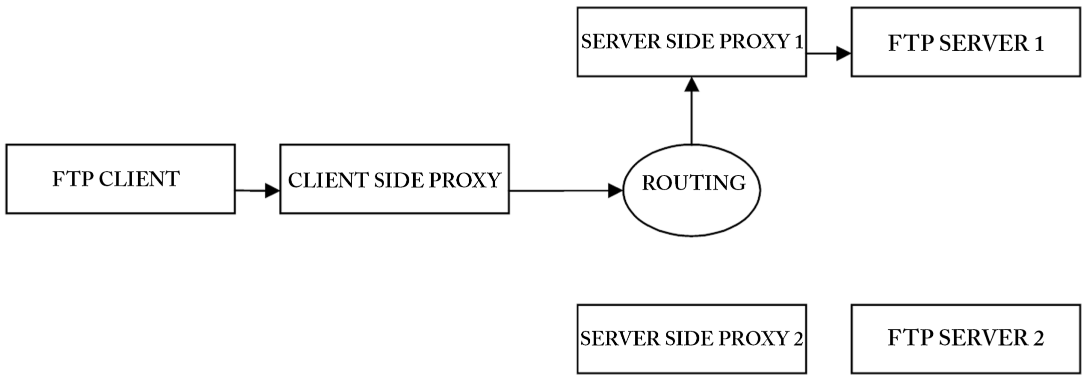


Figure 3.5: client-side proxy routing process

Client-side proxy makes a decision about server-side proxy from the destination address of incoming FTP request i.e. from FTP-server's address.

It initiates a simple routing process. Client-side proxy compares the network address of FTP-server address with the network address stored in the routing table as a text file for routing reference. If the network address of the destination address of FTP-server matches with the network address present in the routing table then client-side proxy connects with the address present next to the matched network address in the routing table.

The comparison process involves conversion of FTP-server address into binary and performing AND operation between the subnet mask of network address stored in the routing table which is also converted to binary, the resultant which is also a network address is compared with the stored network address in the routing table, if both matches then the address present next to the matched entry is fetched for client-side proxy to make a connection with it.

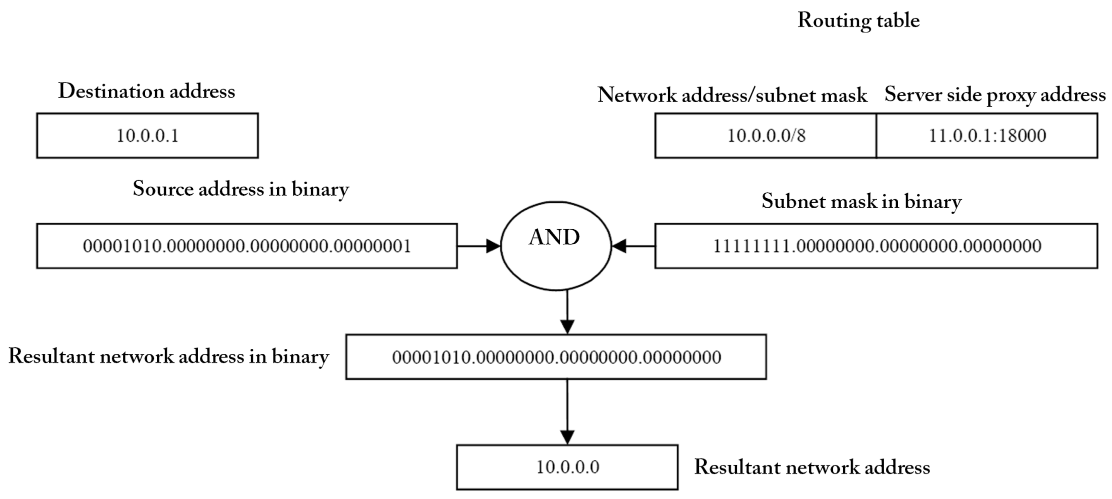


Figure 3.6: Path determination

3.1.4 FTP command reference block

After proxy authentication, client-side proxy routing and FTP authentication, the path from FTP-client to FTP-server is now available via client-side and server-side proxy for sending and receiving commands.

The initial exchange of commands between FTP-client and server is usually getting basic information about the server system like system type, features and modes supported by FTP-server.

Example of one such initial communication scenario between an FTP-client and server is shown below:

Server: Accepted a connection from abc@xyz.net

Client: 220 Welcome to FTP-server

Client: 331 Please specify password.

Server: pass ****

Client: 230 Login successful

Server: SYST

Client: 215 UNIX Type: L8

Server: FEAT

Client: 211-Features:

EPRT

EPSV

MDTM

PASV

RESET STREAM

SIZE

TVFS

211 End

Server: PWD

Client: "/home/user"

In our scenario, all commands that are specified by the FTP-client must pass through the client-side proxy and server-side proxy. Client-side proxy is the first one which is going to be encountered by FTP-client when sending commands to FTP-server as shown in figure 3.7.

Therefore, client-side proxy must be able to understand these FTP commands so that only valid commands passes over the network, rest of the invalid commands should be dropped by the client-side proxy and should not be allowed to travel over the network.

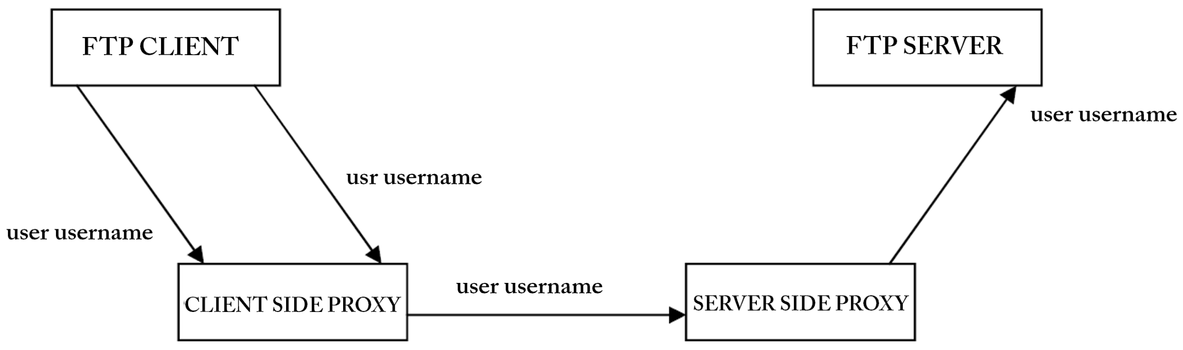


FIGURE 3.7: FTP command reference process

Since the correct command for specifying correct username is "user" not "usr" therefore the command "usr" is dropped and is not allowed to pass over the network and FTP-client is informed about the invalid command input. So that FTP-client may inform the user about invalid command input and prompt the user again to input a correct command.

3.1.5 Control connection setup sequence

The overview of the sequence in which the control connection is established from FTP-client to client-side proxy, client-side proxy to server-side proxy and finally from server-side proxy to FTP-server is shown in figure 3.8. The components involved in the initial connection setup sequence are:

- Proxy authentication block
- Proxy routing block
- FTP authentication block
- Command reference block

When FTP-client wants to transfer files by establishing a connection with FTP-server, which is first encountered by client-side proxy and was waiting and listening on specific IP address and port number to accept an FTP incoming client's request. It means that client-side proxy is also acting like FTP-server, waiting for FTP-client to connect to and process it's request. Processing not only involves forwarding the request to FTP-server but might also process or modify the request at some situations and send the reponse back. However in this case FTP-client is aware of the fact that there is a proxy

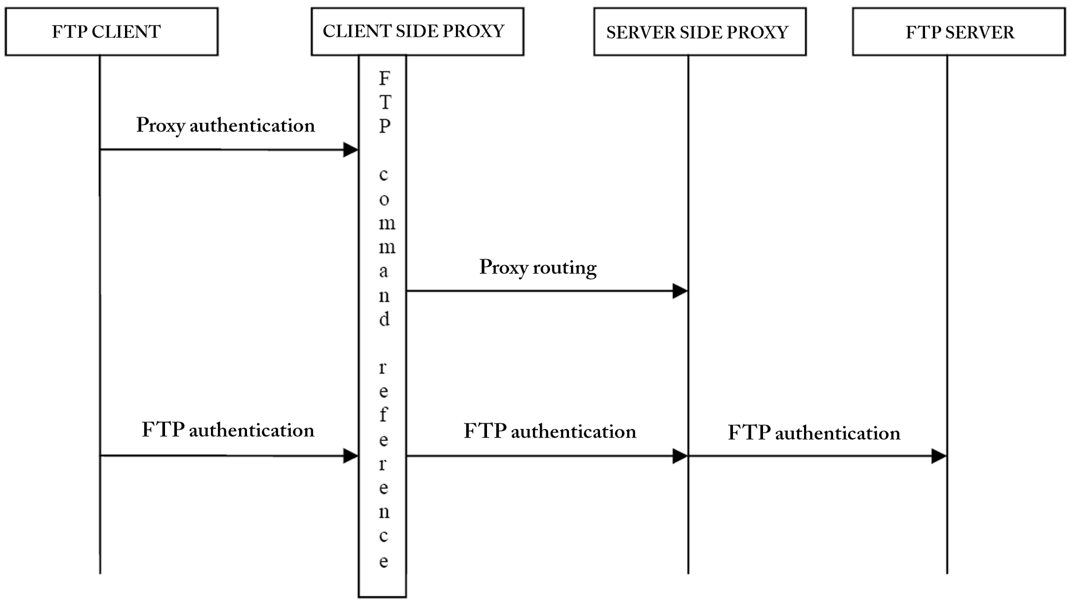


FIGURE 3.8: Control connection setup sequence

mechanism operating in between FTP-client and FTP-server. This is why FTP-client first logs in to the client-side proxy and the authentication process at client-side proxy simply requires a username and password.

Soon after establishing a TCP connection with client-side proxy, it sends a welcome message to FTP-client

$$220 \quad Welcome \quad message \tag{3.4}$$

This welcome message is used by FTP-client to start FTP authentication by sending credentials to client-side proxy.

The number "220" indicates that it is a welcome message, the rest of the message can be altered and makes simply no change in the default processing of FTP-client or server programs. Upon receiving a welcome message from client-side proxy, FTP-client sends its credentials to client-side proxy to get itself authenticated. FTP-client sends username to client-side proxy in the form as in (3.1).

Client-side proxy receives the username from FTP-client and sends a password request "331 Please specify the password" to FTP-client. Then FTP-client sends password to client-side proxy which performs authentication and sends its response back to client-side proxy, in case authentication is successful FTP-client receives successful login message "230 Login successful" from client-side proxy.

After successful authentication of FTP-client by client-side proxy, now client-side proxy will have to make a decision about which server-side proxy to connect to. client-side proxy makes this decision by referring the routing table and comparing the network address present in the routing table with the network address of final destination FTP-server.

If there is match, meaning both network addresses are the same then the entry next to the matched network address is considered as the designated server-side proxy for the specific FTP-client.

Client-side proxy, after authenticating FTP-client sends fake welcome message "220 Welcome to FTP-server" to FTP-client. FTP-client trusts that its from FTP-server and sends credentials for authenticating itself into FTP-server. FTP-client first sends username to client-side proxy trusting client-side proxy would forward it to FTP-server, but the situation behind the wall is different. The username sent to client-side proxy have the syntax as in (3.3).

The syntax of the username is mentioned as such to identify the address of final destination FTP-server [29]. It is forwarded to server-side proxy, server-side proxy extracts the username and address of FTP-server, makes a connection with FTP-server and sends the username to FTP-server as in (3.1).

After receiving username from server-side proxy, FTP-server sends a password request message "331 Please specify the password" to server-side proxy. server-side proxy forwards this password request message to client-side proxy and client-side proxy delivers it to FTP-client.

FTP-client understands the password request message and sends the password to FTP-server via client-side proxy and server-side proxy. FTP sever performs authentication and sends the status of the authentication process to FTP-client via server and client-side proxy respectively. If authentication is successful, FTP-client receives a "230 Login successful" message from FTP-server via server and client-side proxy.

As soon as FTP-client receives successful login message from FTP-server via server-side proxy and client-side proxy, the connection is now setup and can be used to perform signalling between FTP-client and FTP-server. The commands from FTP-client are sent to FTP-server over this connection.

The user protocol interpreter initiates the control connection. The control connection has similar properties like a Telnet protocol. They both send data, username and password over the internet in plain text. FTP uses Telnet protocol on control connection.

This is done by implementing the rules of Telnet procotol into the procedure of connecting the user plane to the server plane into the FTP protocol. It can also be achieved by making use of Telnet protocol to make a control connection between user plane and the server plane [18].

As soon as user initates the connection process, standard FTP commands are exchaged from the user plane to the FTP-server via this control connection.

The control coonection can also be established by some special FTP-client programs that after making a direct connection with FTP-server process sends the commands via this control connection independently without the intervtion of the user and the replies are transmitted over this control conection from the FTP-server plane to user plane [18].

These commands may be used for performing authentication, getting FTP-server system information, setting FTP-server in either active or passive mode and setting the mode in which the data transfer will take place either ASCII or binary. All of these commands are sent over this connection from FTP-client to FTP-server. Also the response or outcome of these commands are sent back to FTP-client via this connection.

This is the only connection which will remain active unless or until either of FTP-client or FTP-server terminates by sending a “QUIT” command or by message “221 Goodbye”. “QUIT” command can be sent either from FTP-client or server program and the responce of the “QUIT” command is “221 Goodbye” message. After receiving “221 Goodbye” message as an acknowledgement of “QUIT” command the connection is closed.

The detailed sequence diagram of initial connection setup sequence from FTP-client to client-side proxy, from client-side proxy to server-side proxy and finally from server-side proxy to FTP-server overall, is shown in figure 3.9.

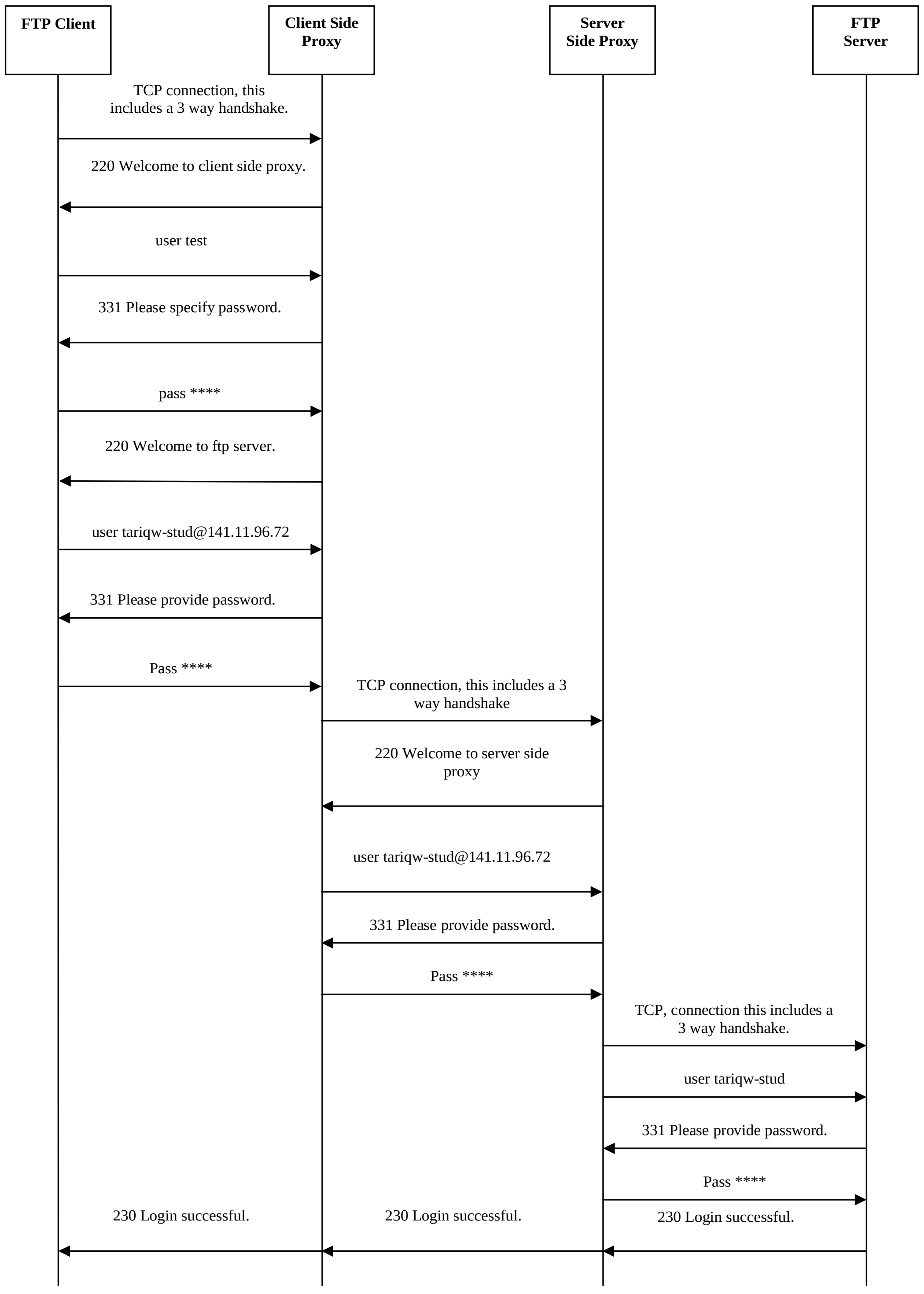
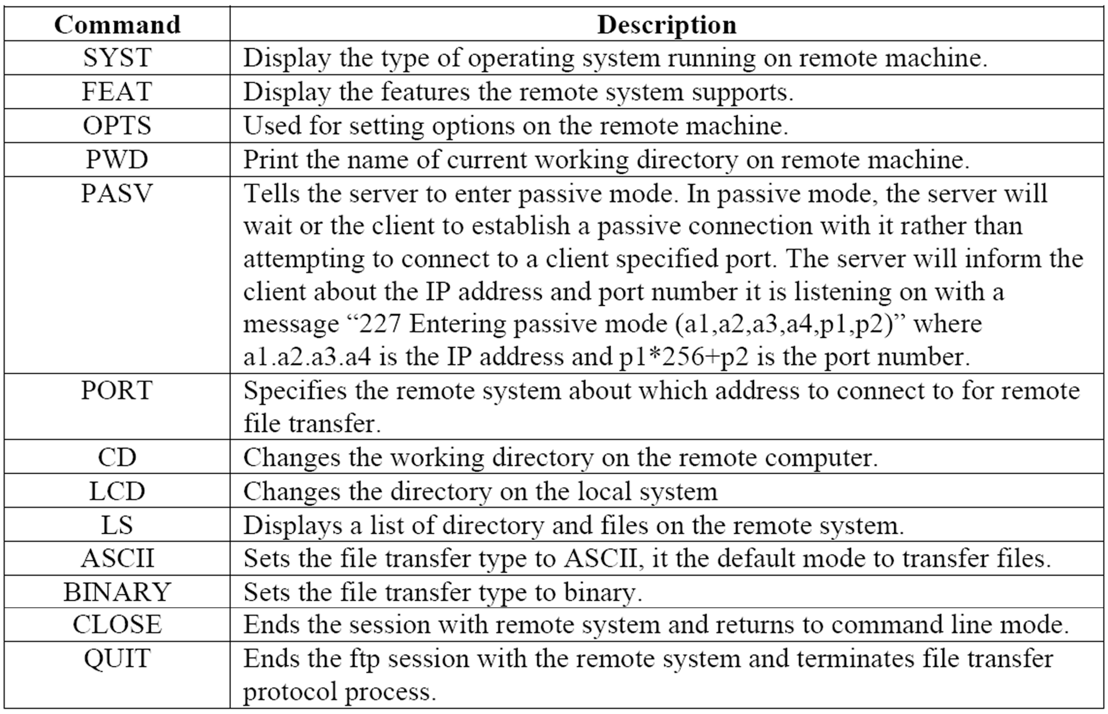

FTP Client
Client Side Proxy
Server Side Proxy
FTP Server
TCP connection, this includes a 3 way handshake.
220 Welcome to client side proxy.
user test
331 Please specify password.
pass ****
220 Welcome to ftp server.
user tariqw-stud@141.11.96.72
331 Please provide password.
Pass ****
TCP connection, this includes a 3 way handshake
220 Welcome to server side proxy
user tariqw-stud@141.11.96.72
331 Please provide password.
Pass ****
TCP, connection this includes a 3 way handshake.
user tariqw-stud
331 Please provide password.
Pass ****
230 Login successful.
230 Login successful.
230 Login successful.

Figure 3.9: Overall control connection setup sequence

3.1.6 FTP initial command sequence

First connection is the main logical TCP connection that is created when FTP session is established. It lasts for the entire FTP session and is used to transfer commands or control information like username, password etc. However, it is not used to transfer data over this connection [31]. After the connection is setup, there are a few common FTP commands a client sends to FTP-server to get information about the FTP-server system capabilities and the features and further commands and modes it supports. Some of the commonly used FTP commands that are used to get information from FTP-server are shown in figure 3.10.



Command	Description
SYST	Display the type of operating system running on remote machine.
FEAT	Display the features the remote system supports.
OPTS	Used for setting options on the remote machine.
PWD	Print the name of current working directory on remote machine.
PASV	Tells the server to enter passive mode. In passive mode, the server will wait or the client to establish a passive connection with it rather than attempting to connect to a client specified port. The server will inform the client about the IP address and port number it is listening on with a message "227 Entering passive mode (a1,a2,a3,a4,p1,p2)" where a1.a2.a3.a4 is the IP address and p1*256+p2 is the port number.
PORT	Specifies the remote system about which address to connect to for remote file transfer.
CD	Changes the working directory on the remote computer.
LCD	Changes the directory on the local system
LS	Displays a list of directory and files on the remote system.
ASCII	Sets the file transfer type to ASCII, it the default mode to transfer files.
BINARY	Sets the file transfer type to binary.
CLOSE	Ends the session with remote system and returns to command line mode.
QUIT	Ends the ftp session with the remote system and terminates file transfer protocol process.

FIGURE 3.10: Basic FTP commands

These commands are passed from FTP-client to FTP-server via client-side and server-side proxy. When "SYST" command is entered on FTP-client it is first passed on to client-side proxy which checks the validity of the command and passes it to server-side proxy and finally, server-side proxy directly delivers it to FTP-server. FTP-server process the request and sends the response of the command, for instance the remote system is UNIX "215 UNIX: L8", to the server-side proxy which forwards it to client-side proxy and finally client-side proxy delivers it to FTP-client. The sequence diagram taken from the FTP-client server communication via client and server-side proxy is shown in figure 3.11.

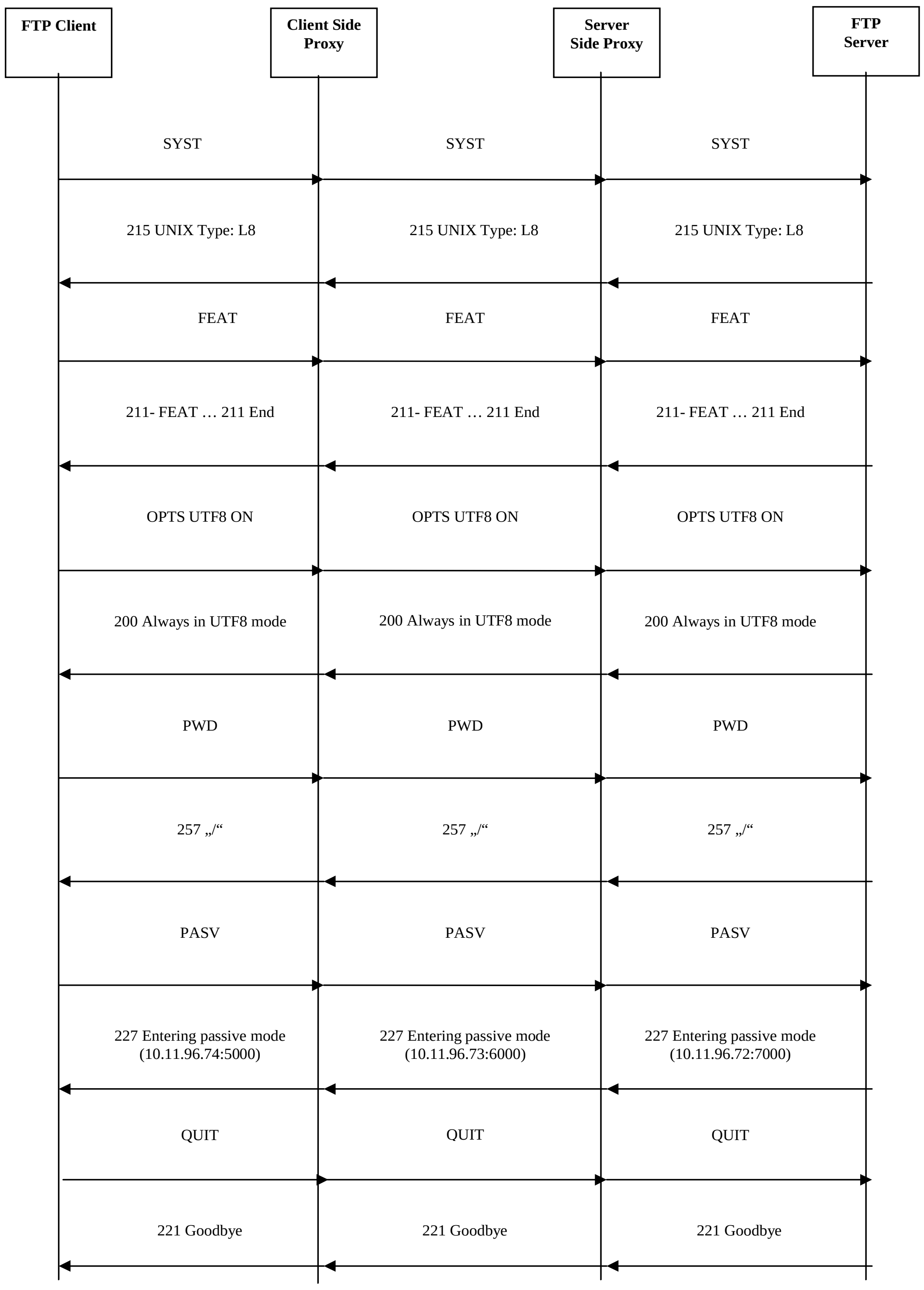


FIGURE 3.11: Initial command sequence

3.2 Data channel block

In order to transfer data from FTP-client to FTP-server or from FTP-server to FTP-client, data connection is to be established first. The earlier connection was used for transferring commands from FTP-client to FTP-server and also receiving their responses. Now the data transfer process requires a separate connection. The first connection for sending control information is control channel and the second connection for transferring data is the data channel. In this case, passive mode is considered for establishing a data channel with FTP-server. FTP-client will make a passive mode connection request to FTP-server via client-side and server-side proxy and the connection is established from FTP-server to FTP-client via serve side and client-side proxy respectively.

The data transfer process involves converting the characters to standard format. For example when transferring from a storage location on one system to storage location on the other system, it is necessary to perform data transformations because data representation on two different systems might be different. For setting similar representation FTP-client can specify the command ASCII to set the data representation type to ASCII [18].

For example in this case a file is requested from FTP-server, FTP-server responds the request by first informing FTP-client about the data type of the requested information and then begins transferring the data.

RETR testfile.dpx

150 Opening Binary mode data connection for testfile.dpx (1025102567 bytes)

ASCII is the default for transferring data and is supported by almost every FTP-client. It is used when both FTP-client and server would like to transfer files to either of them unless specified by either of them to be more convenient with EBCDIC. By doing so, the sender will convert the data from its own representation form to the form requested by the receiving party.

Besides data types FTP also requires the structure of the file to be specified. There are three different types of file structures as specified in [18].

- File-structure
- Record-structure
- Page-structure

File structure has no internal structure and is considered to be a continuous sequence of data byte. In record structure file is considered to be a sequence of records and finally in page structure the file is made up of independent indexed pages. However, file structure is the default to be used when no structure command has been used. It is important to note the structure because it affects transmission mode, interpretation and storage of the file.

3.2.1 Data channel connection sequence

Considering the passive mode for data connection, data connection process initiates when client issues a passive mode command PASV to FTP-server. As soon as FTP-server receives PASV command it responds with a reply “227 Entering passive mode (a1,a2,a3,a4,p1,p2)” along with the IP address and port number where, “a1.a2.a3.a4” is the IP address and “p1*256+p2” is the port number of FTP-server data connection address to connect to.

PASV command is first sent to client-side proxy which forwards it to server-side proxy and server-side proxy finally delivers it to FTP-server. FTP-server process the PASV command and sends back its response by opening a listener on a specific IP address and a random port number.

This specific listener is for FTP-client to connect to and expect to receive data on it. The response “227 Entering passive mode (a1,a2,a3,a4,p1,p2)” is sent to FTP-client via server and client-side proxy respectively.

The passive mode response “227 Entering passive mode (a1,a2,a3,p1,p2)” on the way to FTP-client first reaches server-side proxy, what server-side proxy does is instead of forwarding the passive mode response as it is to client-side proxy, it makes a connection with FTP-server on the address specified in the passive mode response message.

For example, in passive mode response message, the IP address and port number specified by FTP-server is “192,168,10,11,23,112” which is received by server-side proxy. Now server-side proxy instead of forwarding it to client-side proxy makes a direct connection with FTP-server.

Now server-side proxy will send the passive mode response message along with its own IP address and port number for client-side proxy to make a connection with it. For example the passive mode response sent by FTP-server is “227 Entering passive mode (10,1,1,1,25,112)”.

Client-side proxy receives the passive mode response message from server-side proxy and makes a connection with it, modifies the passive mode response message received from server-side proxy to its own IP address and port number and starts listening on it. From the figure shown below client-side proxy sends “227 Entering passive mode (11,1,1,1,150,23)” to FTP-client.

When FTP-client receives a passive mode response message “227 Entering passive mode (11,1,1,1,150,112)” from client-side proxy, FTP-client makes a connection with client-side proxy as shown in figure 3.12.

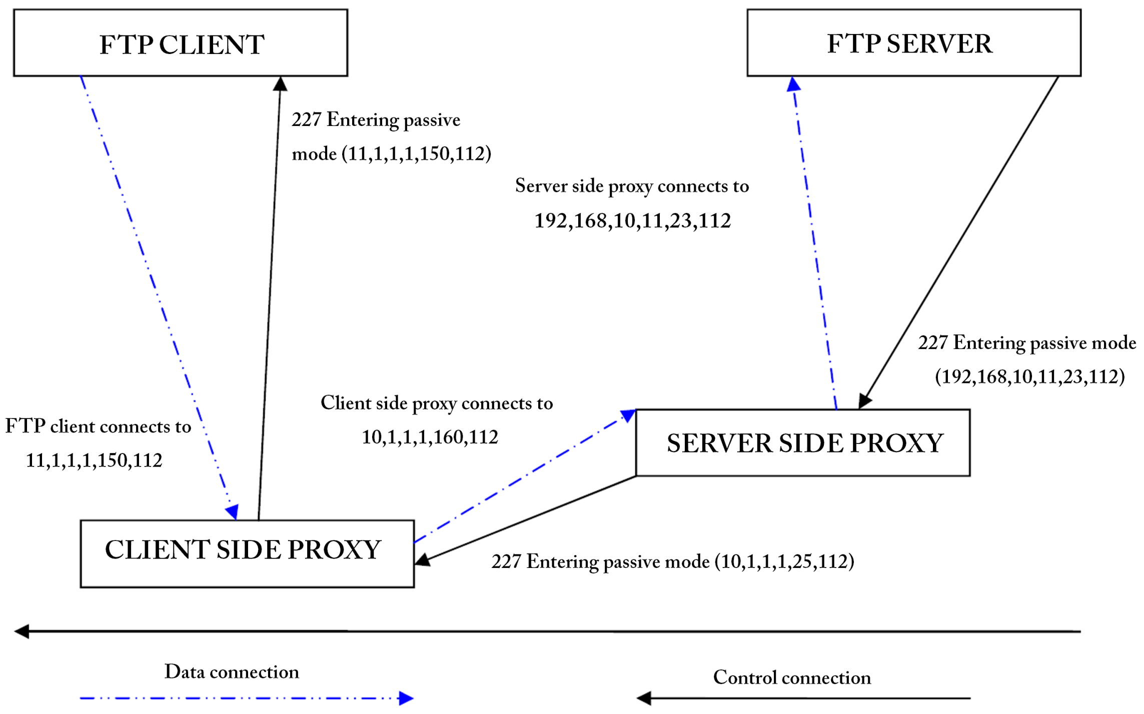

FTP CLIENT
FTP SERVER
227 Entering passive mode (11,1,1,1,150,112)
Server side proxy connects to 192,168,10,11,23,112
227 Entering passive mode (192,168,10,11,23,112)
Client side proxy connects to 10,1,1,1,160,112
FTP client connects to 11,1,1,1,150,112
SERVER SIDE PROXY
CLIENT SIDE PROXY
227 Entering passive mode (10,1,1,1,25,112)
Data connection
Control connection

FIGURE 3.12: PASV command and its responce

Now the connection is established from FTP-client to client-side proxy, from client-side proxy to server-side proxy and finally server-side proxy is also connected to FTP-server. This path will now be used to transfer files from FTP-client to FTP-server or vice versa.

This is how the connection from FTP-client to client-side proxy from client-side proxy to server-side proxy and from server-side proxy to FTP-server is established. Now lets discuss about the sequence it follows to establish this data connection and the role of control channel in establishing this data channel. The sequence diagram for the establishment of data channel also with the role of control channel along with its control information sequence is shown in figure 3.13.

When FTP-client wishes to get a file from remote FTP-server, it first sends a PASV command to FTP-server via client and server-side proxy. The PASV command puts FTP-server in passive mode and enables it to open a connection on one its random port

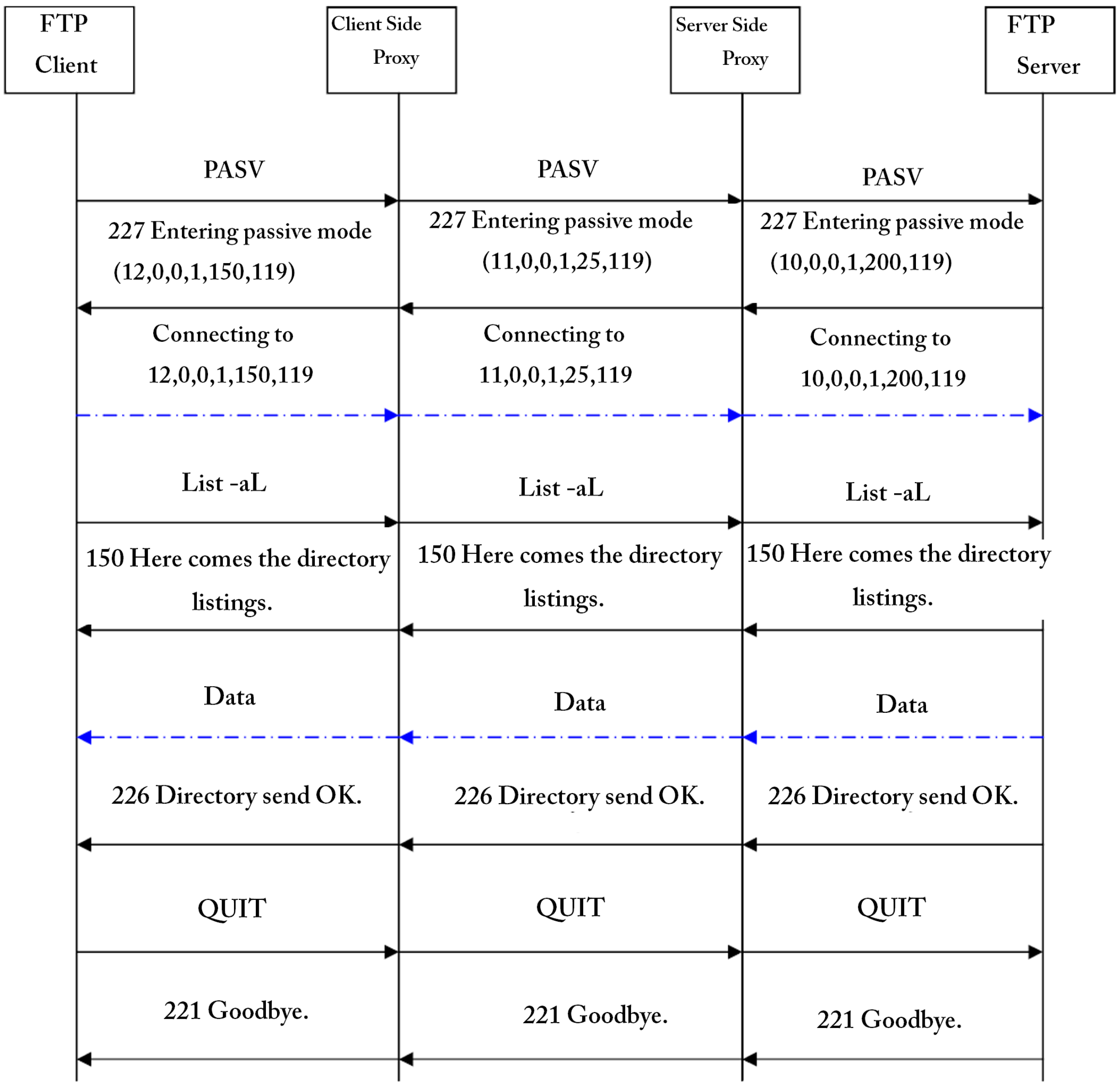

FTP Client
Client Side Proxy
Server Side Proxy
FTP Server
PASV
PASV
PASV
227 Entering passive mode (12,0,0,1,150,119)
227 Entering passive mode (11,0,0,1,25,119)
227 Entering passive mode (10,0,0,1,200,119)
Connecting to 12,0,0,1,150,119
Connecting to 11,0,0,1,25,119
Connecting to 10,0,0,1,200,119
List -aL
List -aL
List -aL
150 Here comes the directory listings.
150 Here comes the directory listings.
150 Here comes the directory listings.
Data
Data
Data
226 Directory send OK.
226 Directory send OK.
226 Directory send OK.
QUIT
QUIT
QUIT
221 Goodbye.
221 Goodbye.
221 Goodbye.

FIGURE 3.13: Data channel connection sequence

from a specified range of ports and sends it to FTP-client. Instead the PASV command response message “227 Entering passive mode (a1,a2,a3,a4,p1,p2)” is first received by server-side proxy which makes a connection with it modifies the passive mode response message with its own IP address and port number and sends it to client-side proxy.

Client-side proxy also performs the same task, makes a connection with the address specified in the passive mode response message i.e. with server-side proxy, modifies the passive mode response message received from server-side proxy with its own IP address and port number and sends it to FTP-client. When FTP-client receives passive mode response message from client-side proxy it makes a connection with it. Finally, the path from FTP-client to FTP-server via client and server-side proxy is now ready to transfer data.

Since FTP-client and client-side proxy are usually in the same network, there might not be that much latency in transferring the data files from client-side proxy to FTP-client or vice versa. Similarly, FTP-server and server-side proxy also usually lies in the same network therefore the latency would not be that much to consider a high data rate transfer mechanism for these connections. That is why TCP type connection is used as a link between FTP-client and client-side proxy and also from server-side proxy to FTP-server as shown in 3.14.

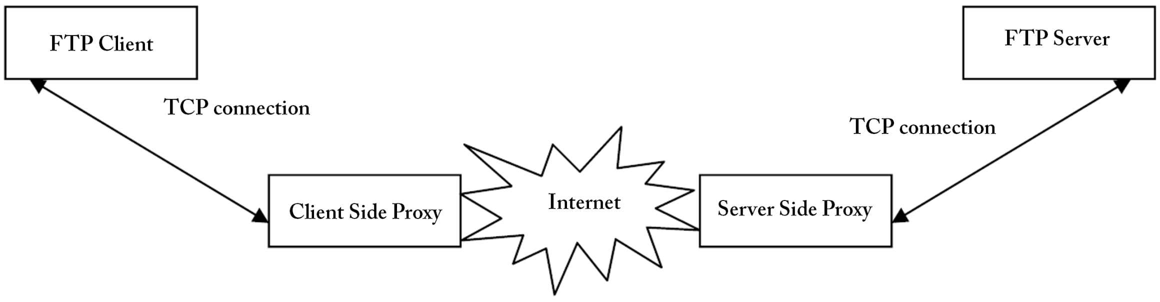


FIGURE 3.14: Gateway scenario

The data from server-side proxy to client-side proxy or from client-side proxy to server-side proxy travels over the Internet and has to face latency and packet loss, due to which the data rate is reduced significantly over this link. Even after the introduction of "RFC 1323 TCP Extensions for High Performance" [32] which was published in 1992 the data rates with latency 70 milli seconds with enhanced window sizes is shown in table 3.1.

Window size	Theoretical max throughput	Realistsic throughput
8 KB	0.9 Mbps	0.8 Mbps
16 KB	1.9 Mbps	1.8 Mbps
32 KB	3.7 Mbps	2-3.5 Mbps
64 KB	7.5 Mbps	3-7 Mbps
128 KB	15 Mbps	6-14 Mbps
256 KB	30 Mbps	10-25 Mbps
512 KB	59.9 Mbps	20-40 Mbps
1 MB	119.8 Mbps	30-60 Mbps
2 MB	239.7 Mbps	60-100 Mbps

TABLE 3.1: **TCP Over WAN Performance Tuning** - Stanislav Shalunov. TCP Over WAN Performance Tuning and Troubleshooting. September 2005

In order to achieve high data rates even in the high latency network environments we need a reliable wide area network protocol faster than TCP.

A transport protocol developed by “Deutsche THOMSON OHG” is RWTP. This protocol has the capability to transfer data at the rate of 5 Gbps per single stream via transcontinental links even in the presence of delay and packet losses.

Therefore we use this protocol as a link between client-side proxy and server-side proxy. Now the path from FTP-client to FTP-server becomes, FTP-client is connected to client-side proxy with TCP, client-side proxy is connected to server-side proxy with RWTP (Reliable Wide Area Network Transport Protocol) and finally server-side proxy is connected to FTP-server with TCP. The diagram for this path is shown in figure 3.15.

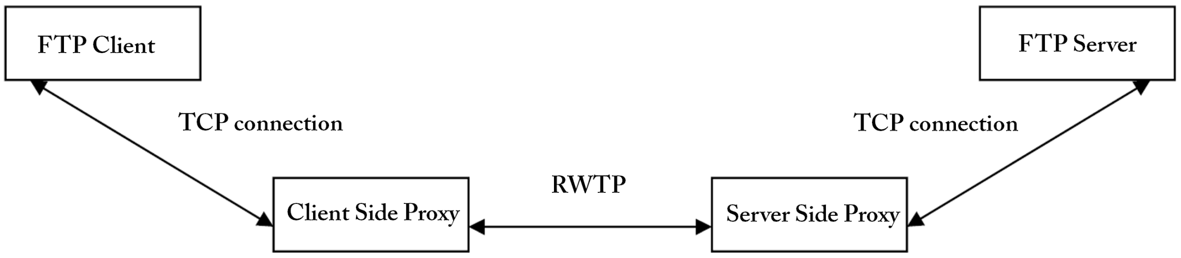


Figure 3.15: Gateway scenario with RWTP

Since now the link is ready for data transfer to and from FTP-client and FTP-server, it can efficiently travel from one end to another. Now the system is ready to be tested over the network under different latency conditions and packet loss rates.

Chapter 4

Implementation

The implementation of proxy application for FTP based data transfer is done in C++ using open source operating system Linux. Testing is performed on Intel(R) Xeon(R) 2.4GHz processors with 8 cores and 8 GBytes RAM. The scenario is implemented on a 1 Gbps Ethernet network, consisting of 1 Gbps switches and 10 Gbps network routers.

C++ “Deutsche THOMSON OHG” proprietary libraries/classes that are used in the implementation process are:

“CIPSocket” library is used because it provides a simple interface for client/server programming. It includes classes for “CTCPSocket” for establishing TCP connections and “CRWTPSocket” for establishing RWTP connections.

“lutils” library includes “CIPAddress” which is used to convert the IP address from string format to “CIPAddress”, it is used as argument in “CTCPSocket” and “CRWTP-Socket”. “CStringUtils” is used for string conversions and “CStringTokenizer” class is used to parse the string, “RETURN TOKEN” and “CLUSTER DELIMITER” are used as argument for “CStringTokenizer”.

The basic requirement of the application to be developed is that it should accept connections from FTP-client, perform its authentication receives address of FTP-server and establish a connection with server-side proxy based on address of FTP-server. Authentication at FTP-server is performed by sending credentials received from FTP-client to FTP-server via client-side and server-side proxy. Upon request of a file to be uploaded or downloaded data channel is created and data is transferred using RWTP protocol which is specially used only on data channel for high-speed data transfer.

The implementation process not only involves performing socket programming for the application but also requires knowledge of object-oriented programming to divide the concepts described in the previous chapter into smaller modules. The division is done such that their functionality is clearly understandable and distinguishable among the other modules. The description of these components is given in an unambiguous way in this chapter.

4.1 Modules

Different modules that take part in C++ implementation process are:

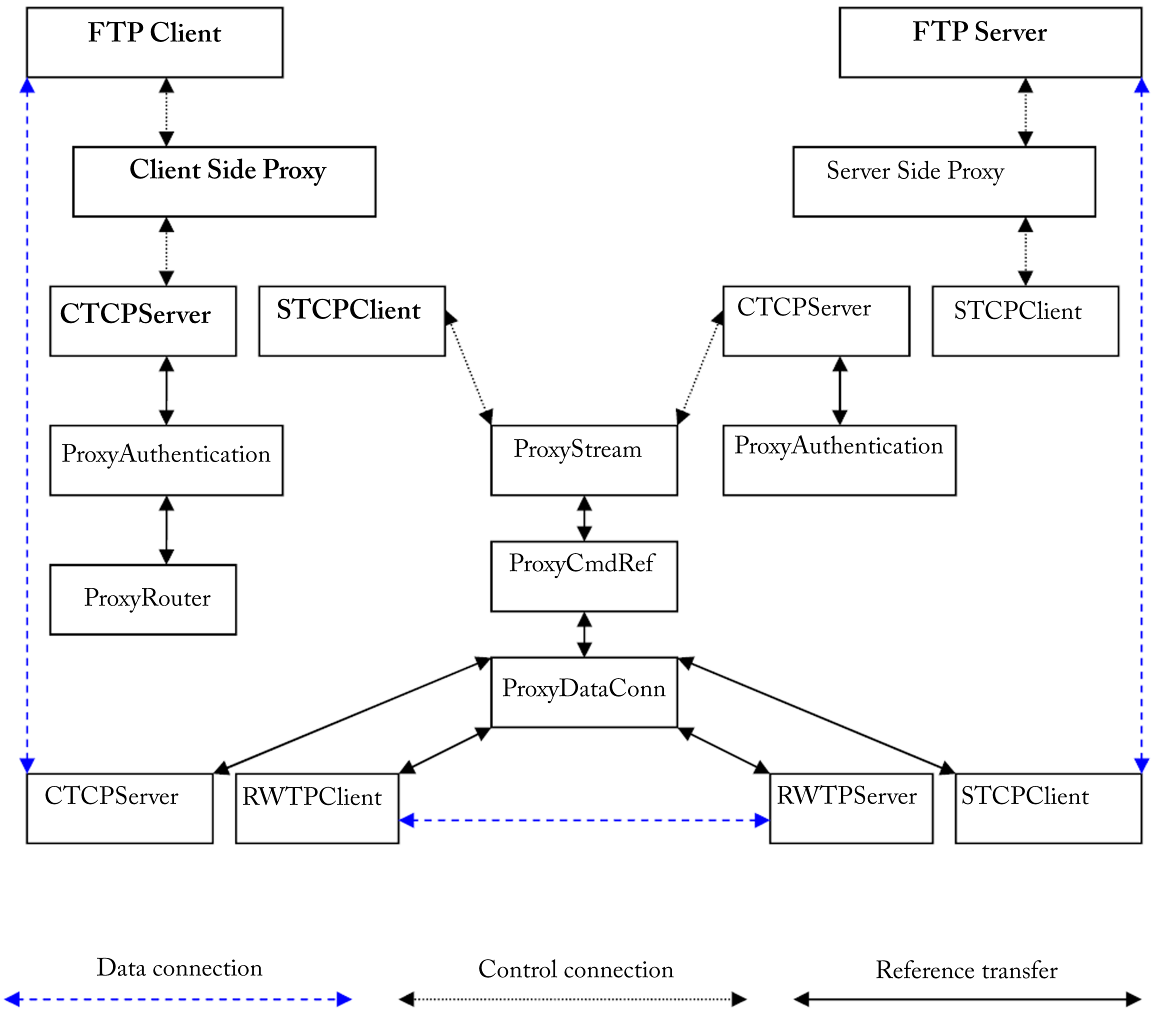


FIGURE 4.1: Implementation model

- CTCPServer: Accepts TCP connections.
- STCPClient: Establishes TCP connections with TCP servers.
- ProxyAuthentication: Performs authentication of FTP-client.

- ProxyRouter: Performs routing and locates server-side proxy.
- ProxyStream: After authentication and locating server-side proxy, it opens a communication stream.
- ProxyCmdRef: Proxy command reference checks the validity of commands.
- ProxyDataConn: Proxy data connection establishes a data connection.
- RWTPClient: RWTP based client. It makes a connection with RWTP server on server-side proxy, this connection is used for data transfer.
- RWTPServer: RWTP based server. It accepts a connection from RWTP client on client-side proxy, this connection is used for data transfer.

4.1.1 ClientSideProxy

The public interface for "ClientSideProxy" module includes a constructor that accepts "CIPAddress" for "ClientSideProxy" module and "Run" method initiates the "ClientSideProxy" execution process. Constructor of "ClientSideProxy" module is used to set the "CIPAddress", type of system which is client-side proxy in this case and concurrency count which indicates concurrency capability of "ClientSideProxy" module or the capability to accept and establish a number of connections. In "Run" method of "ClientSideProxy" module "STCPClient" and "CTCPServer" module are used for accepting and establishing TCP connections. Then "ProxyAuthentication" modules is referenced with "CIPAddress" of ClientSideProxy as argument for authentication of FTP-client at client-side proxy. Then "LocatingSSP" method is called for getting "CIPAddress" of server-side proxy, this process is later discussed in more detail in "ProxyAuthenticaion" and "ProxyRouter" module. "STCPClient" module is called with "CIPAddress" of server-side proxy to make a connection with server-side proxy. "ProxyStream" module is used to open a communication stream between client-side proxy and server-side proxy. Overall, client-side proxy provides the interface for client-side proxy modules.

4.1.2 ServerSideProxy

The public interface for "ServerSideProxy" module includes a constructor that accepts "CIPAddress" for "ServerSideProxy" module and "Run" method initiates the "ServerSideProxy" execution process. Constructor of "ServerSideProxy" module is used to set the "CIPAddress", type of system which is server-side proxy in this case and concurrency count which indicates concurrent capability of "ServerSideProxy" module or the capability to accept and establish a number of connections simultaneously. In "Run" method of “ServerSideProxy” module “STCPClient” and “CTCPServer” module are used for accepting and establishing TCP connections. “LocatingFTPServer” method is accessed from “ProxyAuthentication” module which returns the “CIPAddress” of FTP-server. Then “STCPClient” module is accessed with “CIPAddress” of FTP-server to make a connection with FTP-server. “FTPServerAuthentication” method from “ProxyAuthentication” module is accessed which receives the authentication status from FTP-server. “ProxyStream” module with “CIPAddress” of server-side proxy as well as FTP-server is used to open a communication stream between server-side proxy and FTP-server. Overall, server-side proxy provides the interface for server-side proxy modules.

4.1.3 CTCPServer

The public interface for “CTCPServer” includes a constructor, “CTCPServer init” and “CTCPListen” method. Constructor is used to set it as a listening socket and puts into singleton mode. “CTCPServer Init” method accepts “CIPAddress” as argument. client-side TCP server is a TCP server that waits for connections from TCP client. “CTCPSocket” is used for accepting TCP connections. If “Accept” call fails then a value is returned from “CTCPServer” identifying failed “Accept” with logged error message. Otherwise, “CTCPSocket” is returned.

4.1.4 STCPClient

The public interface for “STCPClient” includes a constructor, “STCPClient Init” and “STCPConnect” method. Constructor is used to set it as a non-listening socket. “STCPClient Init” accepts “CIPAddress” as argument. Server-side TCP client is a TCP client that makes a connection with TCP server. “CTCPSocket” is used in this case for making a connection with TCP server. If “Connect” call fails then a value is returned from “STCPClient” identifying failed “Connect” with logged error message. Otherwise, “CTCPSocket” is returned. Scenario is shown in figure 4.2.

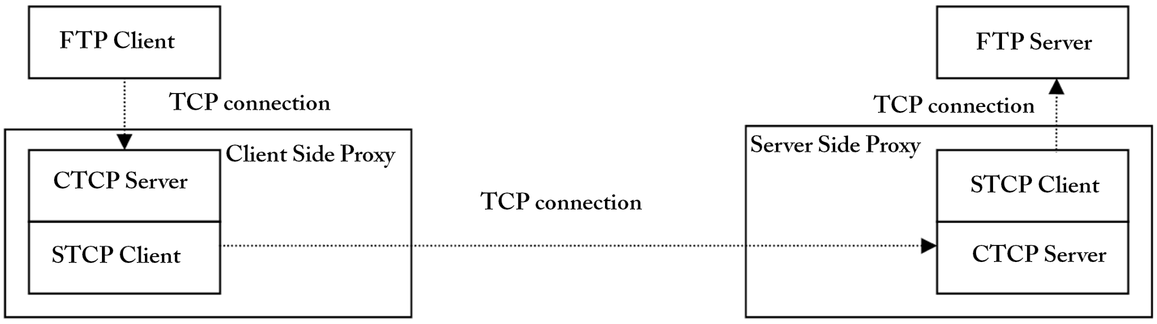


FIGURE 4.2: TCP WAN connection

4.1.5 ProxyAuthentication

The public interface for "ProxyAuthentication" module includes a constructor, "CspAuthentication", "LocatingSsp", "SspAuthentication", "LocatingFTPServer" and "FTPServer-Authentication" methods. "CspAuthentication" accepts "CTCPSocket" of client-side proxy it sends a "220 Welcome message" to FTP-client and receives username from it in a receive buffer using "Recv" system call from "CIPSocket" library, after receiving username, "331 Please specify password" request is sent to FTP-client. It then responds with password. Proxy authentication module then responds with status of authentication. If authentication is unsuccessful "530 Incorrect login" is sent with a false boolean as return value, otherwise "230 Login successful" message is sent with a true return boolean from "CspAuthentication" module.

"LocatingSsp" module sends another "220 Welcome message" to FTP-client. This "220 Welcome message" makes FTP-client to give away its credentials for FTP-server. This is going to help in making a decesion about which server-side proxy to rely on by referencing so called routing table from "ProxyRouter" module. FTP-client sends username for FTP-server in the form as shown below:

user username@ipaddress

Where "user" is the command and "ipaddress" is the address of final destination FTP-server. IP address of FTP-server is extracted, port number is appended at the end on which FTP-server is listening on. Then proxy authentication module sends "331 Please specify password" request to FTP-client, upon receiving it FTP-client replies with password. Username and password are extracted and stored into a buffer for future references. "ProxyRouter" module is referenced with argument "CIPAddress" of FTP-server to get "CIPAddress" of server-side proxy.

“SspAuthentication” method accepts “CTCPSocket” as argument and sends username received in “LocatingSsp” method to server-side proxy and upon receiving “331 Please specify password”, password is also sent to server-side proxy module. After that “SspAuthentication” waits for authentication status from server-side proxy. If status of authentication from server-side proxy about FTP-server is unsuccessful then a value is returned identifying unsuccessful authentication with logged error message.

“LocatingFTPServer” accepts “CTCPSocket” of client-side proxy and receives username from client-side proxy sent from “SspAuthentication” method, the username is extracted separately and IP address is also extracted along with port number. “CIPAddress” of FTP-server is returned.

“FTPServerAuthentication” method accepts “CTCPSocket” of FTP-server and sends username as shown below:

user username

It was received from client-side proxy in method “LocatingFTPServer” and waits for a message from FTP-server, after receiving “331 Please specify password” server-side proxy sends password to FTP-server and waits for authentication status using “Recv” call, if authentication is unsuccessful, it is indicated by a false boolean with a logged error message. Otherwise a true boolean is returned indicating successful login.

4.1.6 ProxyStream

The public interface for “ProxyStream” includes a constructor, “Proxy Init” and “ProxyStream Run” methods. “ProxyStream” constructor sets “RETURN TOKENS” and “CLUSTER DELIMITERS”. “Proxy Init” accepts two “CTCPSocket” and the type of the system which is either client-side proxy or server-side proxy.

“ProxyStream Run” method on client-side proxy when receives a message from FTP-client it passes the reference to proxy command reference module “ProxyCmdRef” depending upon the return value from proxy command reference module identifying the validity of the command it is forwarded to server-side proxy otherwise if the value identifying an invalid command the command is not forwarded to server-side proxy.

When it receives a message from server-side proxy, it checks for PASV mode response message and if it is PASV mode response message it transfers the reference to proxy data connection module. After receiving the reference back from proxy data connection module it modifies the PASV mode response message with its own IP address and forwards it to FTP-client.

Similarly, proxy stream on server-side proxy when receives a message from FTP-server it checks for PASV mode response message and if it is PASV mode responses message it transfers the reference to proxy command reference module. After receiving the reference back from proxy command reference module it modifies the PASV mode response message with its own IP address and forwards it to client-side proxy.

After receiving an identification for “QUIT” or “221 Goodbye” message from “ProxyCmdRef” module, “CTCPSocket” are closed and the module returns successfully.

4.1.7 ProxyCmdRef

The public interface for Proxy command reference “ProxyCmdRef” module includes a constructor and “CmdRef” method. Constructor sets “RETURN TOKENS” and “CLUSTER DELIMITERS”. “CmdRef” method accepts commands and system status as argument. System status defines the type of system whether it is a client-side proxy or a server-side proxy and commands are FTP based. “CmdRef” method checks for valid commands and passes them if they are valid otherwise invalid commands are not passed over the network. By doing so network will be saved from the extra overhead of transmitting invalid commands and their responces all the way back from FTP-server informing that the command was invalid.

“CmdRef” method receives command as argument, it checks for the validity of that command by comparing previously stored basic set of FTP commands with extracted first part of command, if it is valid i.e. present in the stored set of basic commands an integer value is returned identifying the validity of the command. On the other hand if the command is invalid an integer value is returned identifying the invalid command.

It also checks for responses of commands, if the command received is a PASV mode response message, reference is passed further onto proxy data connection module “ProxyDataConn” along with the response message and system status.

4.1.8 RWTPClient

The public interface for “RWTPClient” includes a constructor, “RWTPClient Init” and “RWTPConnect” method. Constructor sets into a non-listening socket. “RWTPClient Init” accepts “CIPAddress” as argument. “RWTPConnect” creates a non-listening “CRWTPSocket”, sets send and receive buffer sizes to 16 MBytes, packet size to 8800 bytes and data rate is set to 10 Gbits/second. It is used at client-side proxy to establish a connection with server-side proxy. If “Connect” call fails then a value is returned from “RWTPClient” identifying failed “Connect” with logged error message. Otherwise, “CRWTPSocket” is returned.

4.1.9 RWTPServer

The public interface for “RWTPServer” includes a constructor, “RWTPServer Init” and “RWTPListen” method. Constructor sets it as a listening socket and puts it into singleton mode. “RWTPServer Init” method accepts “CIPAddress” as argument. “RWTPListen” creates a listening “CRWTPSocket”, sets send and receive buffer sizes to 16 MBytes, packet size to 8800 bytes and data rate is set to 10 Gbits/second. RWTP based server accepts “CIPAddress” type IP address, creates “RWTPSocket” and starts listening on it. Also, it accepts a RWTP based connection. It is used at server-side proxy to accept a connection request from client-side proxy. If “Accept” call fails then a value is returned from “CTCPServer” identifying failed “Accept” with logged error message. Otherwise, “CRWTPSocket” is returned as. Scenario is shown in figure 4.3.

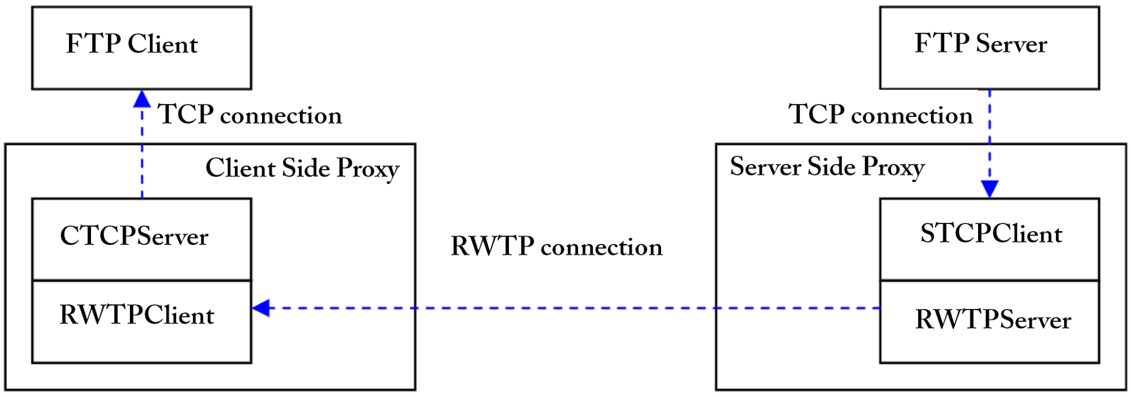


FIGURE 4.3: RWTP WAN connection

4.1.10 ProxyDataConnection

The public interface for “ProxyDataConn” includes a constructor, “Dataconn Init”, “ServerSideProxyDataConn” and “ClientSideProxyDataConn” methods. Constructor sets “RETURN TOKENS” and “CLUSTER DELIMITERS”. “Dataconn Init” method accepts command and system status as argument. “Dataconn Init” accepts passive mode response message and the type of the system, either client-side proxy or server-side proxy. It extracts the IP address and port number from passive mode response message

227 Entering passive mode (192,168,1,1,23,50)

The IP address is extracted, each octet of IP address is extracted as string and is appended with dotted notation to make a complete 32 bit IP address. Port number is also extracted as string but is then also converted to integer so that first octet is multiplied with "256" and then added with the second octet to get a complete 16 bit port number. The IP address along with port number in the form "192.168.1.1:5938" is used to get "CIPAddress" type address for "CIPSockets".

Type of the system is used to check whether it is a client-side proxy or server-side proxy. In case of server-side proxy, the connection is made with the address of type "CIPAddress" by making a "CTCPSocket" and establishing a connection with it which is FTP-server. After that IP address of server-side proxy is fetched from the text file "ServerSideProxyAddress.txt", first octet of port number is fixed at "25" second port number is randomly generated to keep the range of port number being used for this WAN connection in the range of 6400-6655 and finally appended together and is used with IP address to get "CIPAddress" which is used by "ServerSideProxyDataConn" method to start listening on "CRWTPSocket" for client-side proxy to connect to.

Similarly in case of client-side proxy, "ClientSideProxyDataConn" method, "CRWTP-Socket" connection is established with the IP address received from "Dataconn Init" method. After that client-side proxy fetches its IP address from "ClientSideProxyAddress.txt", first octet of port number is fixed at "24" and the second octet is randomly generated to keep the port range used for connection with FTP-client limited in the range of 6144-6399 and finally appended together and is used with IP address to get "CIPAddress" which is used by "ClientSideProxyDataConn" to start listening on "CTCPSocket" for FTP-client to connect to.

server-side proxy after establishing successful connections with FTP-server and client-side proxy, server-side proxy sends and receives in a loop until EOF is reached and sockets are closed. Similarly, client-side proxy after establishing successful connections with server-side proxy and FTP-client, client-side proxy sends and receives in a loop until EOF is reached and sockets are closed.

The send and receive process links the two different connection types RWTP and TCP by receiving the data on "CTCPSocket" and sending it to "CRWTPSocket".

4.1.11 ProxyRouter

The public interface for "ProxyRouter" includes a constructor and "ReferRoutingTable" method. "ReferRoutingTable" accepts "CIPAddress" as argument which is the destination address of FTP-server in our case. It fetches the network address and its subnet mask present in the routing table configuration file "ProxyRoutingTable.txt". It converts the received destination address of FTP-server to binary along with the subnet mask from the routing table file. After converting them to binary, an AND operation is performed between the destination address and subnet mask. This operation gives the network address of the destination address. It is then converted back to dotted decimal notation. Then the resultant network address is compared with the network address stored in the "ProxyRoutingTable.txt" file, if there is a match then address next to the matched network address which is the address of server-side proxy is fetched from the "ProxyRoutingTable.txt" and returned. By doing so routing mechanism is tested on this smaller scale testbed which generalizes its concept and can also be used on a large network with several routers.

4.2 Flow chart: client-side proxy

Client-side proxy starts by first accepting a connection from FTP-client. Then FTP-client provides credentials to client-side proxy for authentication. If authentication is successful, client-side proxy locates the address of server-side proxy, otherwise connection is terminated.

After locating server-side proxy, client-side proxy establishes a connection with it, sends credentials of FTP-client to log into FTP-server. client-side proxy waits for authentication status message from server-side proxy. If authentication into FTP-server is not successful then FTP-client is informed about the status and connection is terminated. Otherwise, a connection is setup between FTP-client and FTP-server via client-side and server-side proxy.

The commands that pass through client-side proxy from FTP-client are checked by client-side proxy whether the command being passed is valid or not. If the command is not valid it is dropped and is not sent over the network otherwise command is sent to server-side proxy.

The response of command received from server-side proxy is also checked for passive mode response, if there is a passive mode response message from server-side proxy, client-side proxy makes a connection with it, modifies the passive mode response message with its own address and sends it to FTP-client and starts listening on the address sent to it.

After establishing a successful connection with server-side proxy and accepting a connection from FTP-client it receives data from either direction, i.e. from FTP-client or from server-side proxy. At end of file the connection is closed and the thread is terminated. If client-side proxy receives a command "QUIT" from FTP-client, it sends it to server-side proxy and terminates the connection. The connection is also terminated when client-side proxy receives "221 Goodbye" message from server-side proxy as shown in figure 4.4.

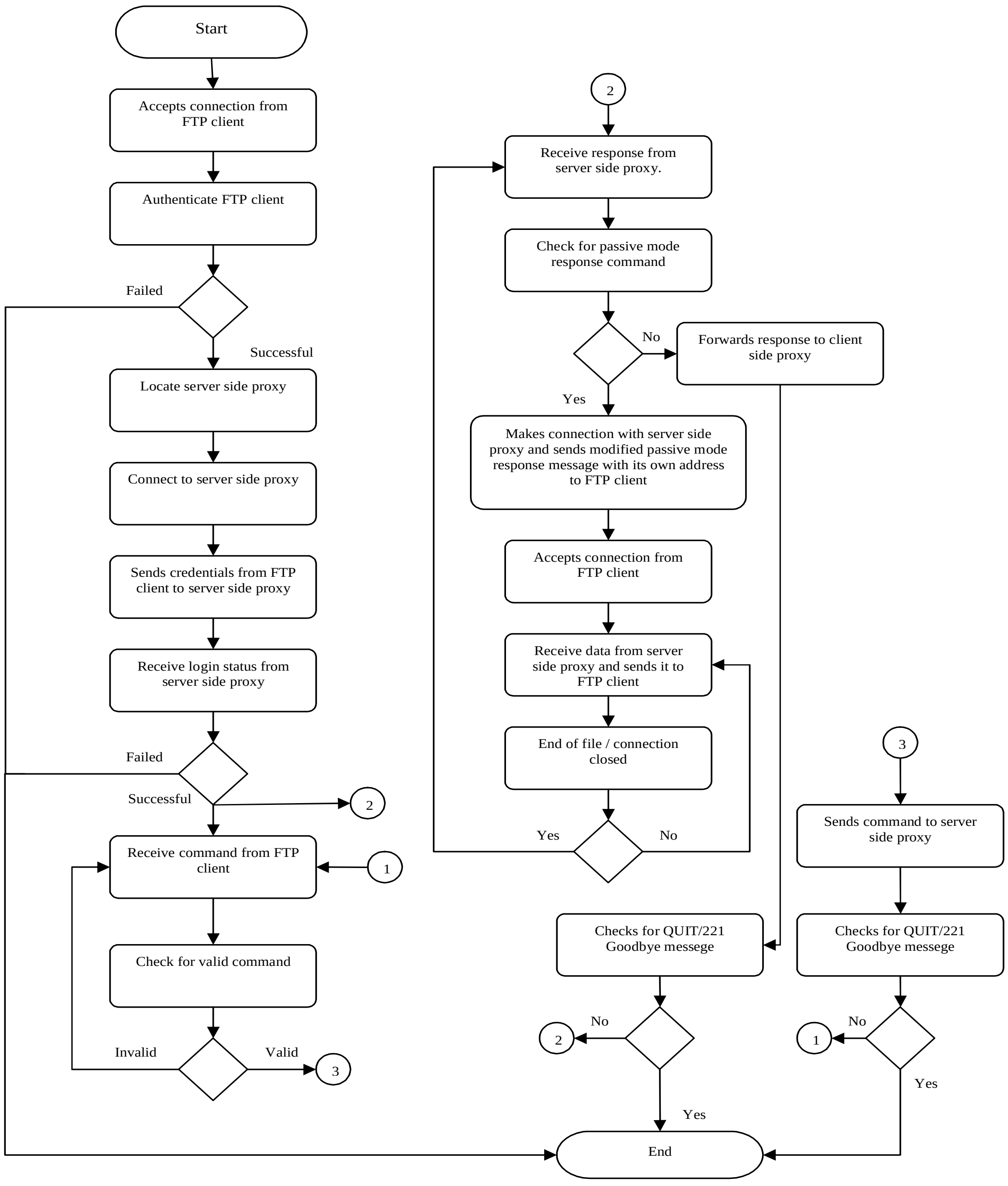

Start
Accepts connection from FTP client
Authenticate FTP client
Failed
Successful
Locate server side proxy
Connect to server side proxy
Sends credentials from FTP client to server side proxy
Receive login status from server side proxy
Failed
Successful
2
Receive command from FTP client
1
Check for valid command
Invalid
Valid
3
2
Receive response from server side proxy.
Check for passive mode response command
No
Forwards response to client side proxy
Yes
Makes connection with server side proxy and sends modified passive mode response message with its own address to FTP client
Accepts connection from FTP client
Receive data from server side proxy and sends it to FTP client
End of file / connection closed
Yes
No
Checks for QUIT/221 Goodbye messege
No
2
Yes
3
Sends command to server side proxy
Checks for QUIT/221 Goodbye messege
No
1
Yes
End

FIGURE 4.4: Flow chart: client-side proxy

4.3 Flow chart: server-side proxy

server-side proxy accepts a connection from client-side proxy and receives credentials from it. After that it extracts address of final destination FTP-server from the credentials and makes a connection with FTP-server.

After making a successful connection with FTP-server, server-side proxy sends credentials received from client-side proxy to FTP-server for authentication. If authentication is successful, server-side proxy opens a communication stream between client-side proxy and FTP-server.

Server-side proxy receives commands from client-side proxy and forwards it to FTP-server, it also checks for “QUIT” command from client-side proxy. If it receives a “QUIT” command or “221 Goodbye” message from client-side proxy it terminates. Otherwise it will continue receiving commands from client-side proxy and forwarding it to FTP-server.

It also receives the response of commands from FTP-server. It checks whether the response is a passive mode response or not. If it is a passive mode response then it makes a connection with the address specified in the passive mode response message, modifies it with its own address and forwards it to client-side proxy and waits for it to establish a connection with it. Otherwise if its not a passive mode response message it is forwarded to client-side proxy.

The connection established from passive mode response message is used for transfering data, as soon as data is completely transferred server closes the data connection and continue receiving response of commands from FTP-server as shown in figure 4.5.

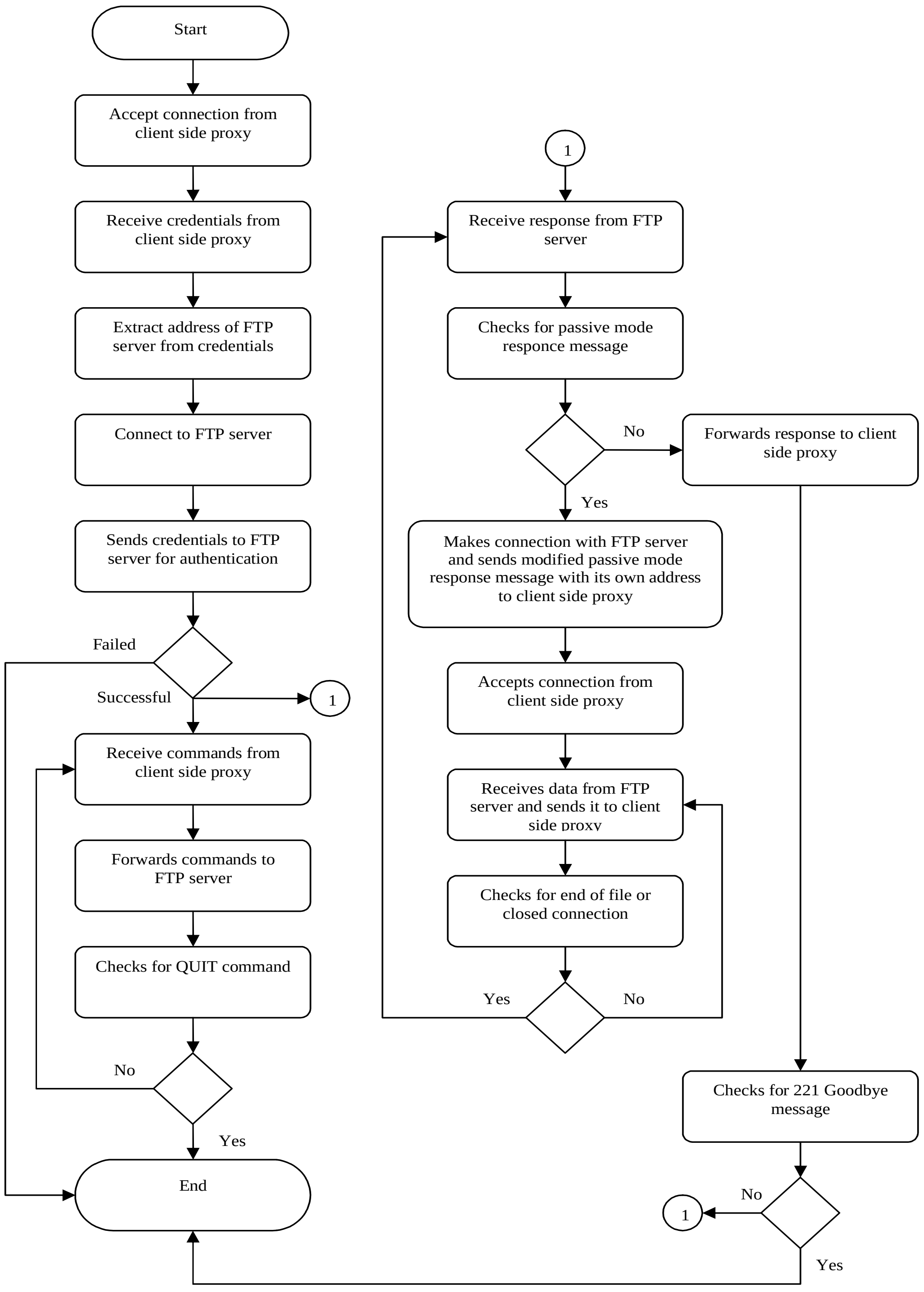

Start
Accept connection from client side proxy
Receive credentials from client side proxy
Extract address of FTP server from credentials
Connect to FTP server
Sends credentials to FTP server for authentication
Failed
Successful
1
Receive commands from client side proxy
Forwards commands to FTP server
Checks for QUIT command
No
Yes
End
1
Receive response from FTP server
Checks for passive mode responce message
No
Forwards response to client side proxy
Yes
Makes connection with FTP server and sends modified passive mode response message with its own address to client side proxy
Accepts connection from client side proxy
Receives data from FTP server and sends it to client side proxy
Checks for end of file or closed connection
Yes
No
Checks for 221 Goodbye message
No
1
Yes

Figure 4.5: Flow chart: server-side proxy

Chapter 5

Performance Evaluation

In this chapter, performance test of the proxy scenario is performed. This is achieved with a network emulator to evaluate the system model against delay and packet losses. In this scenario FTP-client and client-side proxy or server-side proxy and FTP-server belong to the same local network with a link speed of 1 Gbps. The connection from each proxy towards the WAN is using 10 Gbps full-duplex fiber link. The proxies are interconnected by using 10 Gbps routers (the cloud) as shown in the figure 5.1. In this way each proxy has two network interface cards. One of them is used to connect the proxy over the network and the other is used to connect either to the FTP-client in case of client-side proxy or to the FTP-server in case of server-side proxy.

Both client-side proxy and server-side proxy are running on high-end Linux systems. Each system has 4 processors of 3 GHz and RAM of 8 GBytes.

Tests are performed to compare the performance of using TCP and RWTP for data transfer with varying delay levels on a range of 6 up to 200 ms (milli seconds) with packet loss rate of 0.1 percent and 0.3 percent. In the first phase, link utilization is measured by using Iperf tool with no delay and no packet loss to check the maximum level that the link can provide. Iperf tool is also used to show the performance a direct TCP connection in transfering raw data or using FTP in the presense of delay and packet losses.

In the second phase, the performance of indirect connection using TCP as well as RWTP is evaluated.

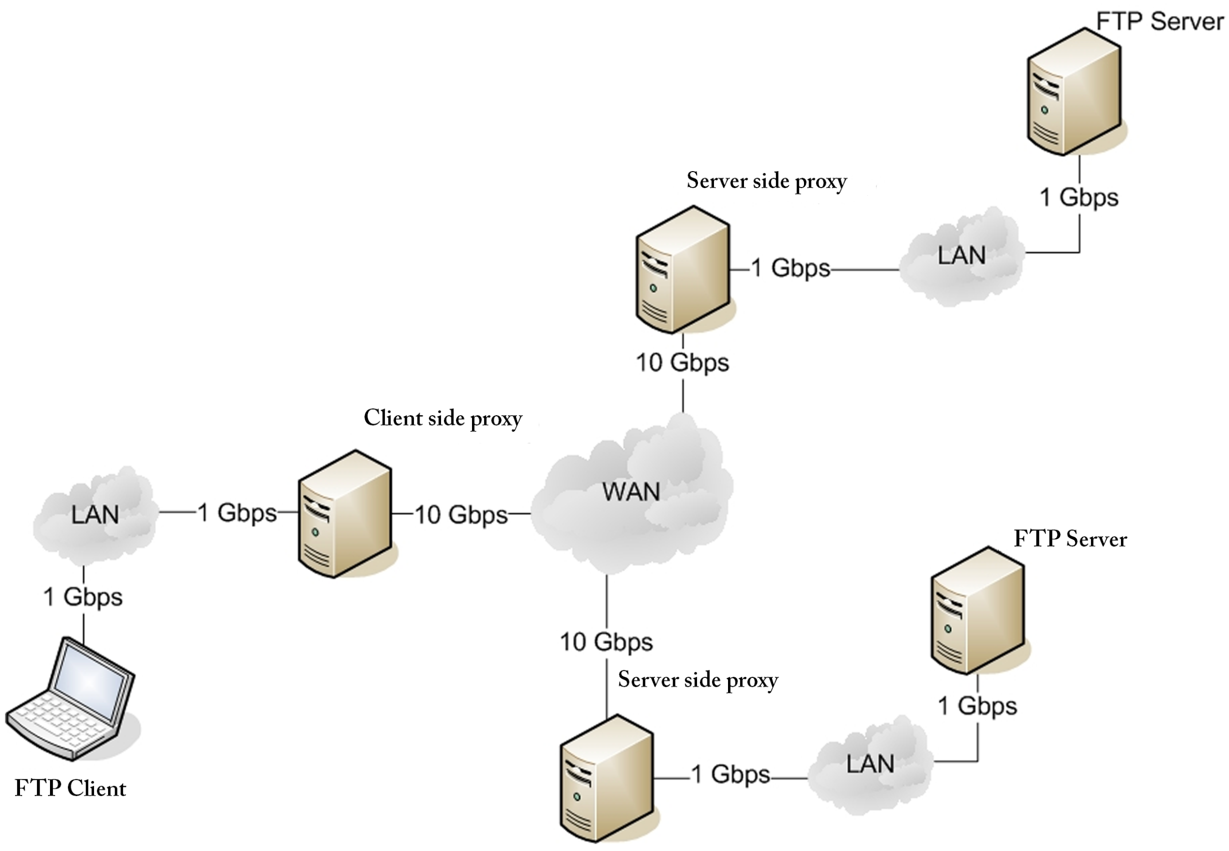
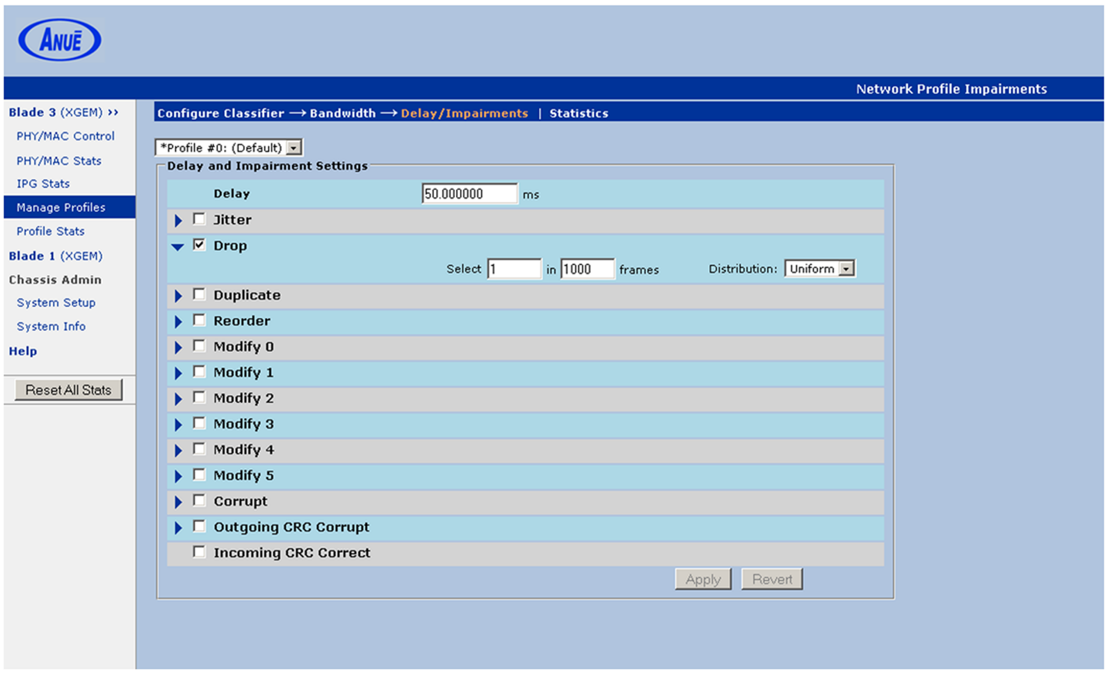


FIGURE 5.1: The network model used for performance evaluation

5.1 Network Emulator

Network emulation can be achieved by introducing a device on the network that alters the packet flow in a way that reproduces closely the behaviour of the network equipment on traffic [33]. The device can be either a computer running a software on it that alters packets as they passes through, or it can also be a specific device dedicated to perform emulation process. Variety of attributes possesed by a network emulator are latency, bandwidth, packet loss and jitter.

The Anue Network Emulator is a hardware tool that simulates real world network conditions for tests and validation of network-based products and solutions prior to their deployment. Anue XGEM Advanced Ethernet Network Emulators are ideal for precisely simulating WAN using 10 Gbps Ethernet interfaces in a lab environment [34].
]



*Profile #0: (Default)

Delay 50.000000 ms

☑ Drop

Select 1 in 1000 frames Distribution: Uniform

FIGURE 5.2: 10 Gbps Network Emulator

5.2 TCP window size adjustment

TCP is the most commonly used protocol on the Internet for reliable, connection oriented, window based data transfer. TCP window size is usually equal to the TCP send and receive buffer size. Any mismatch might lead to problems like receive buffer is too small to store the received data [35]. In traditional TCP the receiving window size cannot be larger than 64 KBytes because the header field that is used to tell about the window size is 16 bits long. The theoretical maximum throughput supported by having a window size of 64 KBytes with a RTT delay of 100 ms.

$$Windowsize/RTT = 64KBytes/100milliseconds = 5Mbps \tag{5.1}$$

After the introduction of “RFC 1321 on TCP Extensions for High Performance”, TCP window size can be increased to 1 GBytes [32]. By increasing the TCP window size to 16 MBytes the theoretical maximum throughput with a RTT delay of 100 ms becomes.

$$Windowsize/RTT = 16MBytes/100milliseconds = 1.3Gbps \tag{5.2}$$

TCP send and receive buffer sizes can be modified by changing the default values in the kernel. Increasing TCP transmit and receive buffer sizes, “tcp rmem” and “tcp wmem” respectively may give tremendous increase in throughput on high bandwidth networks [36]. The optimal buffer sizes can be calculated as:

$$buffersize = Bandwidth * RTT \tag{5.3}$$

For example the available bandwidth is 100 Mbps and a RTT of 50 ms the buffer size is about 610 KBytes, “tcp rmem” and “tcp wmem” should be set to 610 KBytes.

Modifying “tcp rmem” tells the kernel about the receive buffer size parameters and similarly for the “tcp wmem” which tells the kernel about transmit buffer size parameters per TCP connection. When changing the buffer size there are three variables, the first value informs the kernel about the minimum receive or transmit buffer, the second value informs the kernel about the default receive or transmit buffer and the third value informs the kernel about the maximum receive or transmit buffer for each TCP connection. According to the above calculations the window size will be modified to be 16 MBytes for send and receive buffers. With this buffer size and the available 1 Gbps bandwidth the performance will be evaluated with a RTT of 100 ms.

5.3 Measuring TCP bandwidth performance with Iperf

“Iperf” is a tool to measure wired and wireless link bandwidth and throughput. It is used to check the health of the link by making end to end status measurements using TCP or UDP. It is available as open source for both Windows and Linux systems. Typical scenario for performing bandwidth measurement using “Iperf”, is running a server on one system and a client on another system. When the two systems are interconnected by a network the client side of Iperf will measure the throughput of the received data from the server side.

In Linux based system the Iperf server is started by the command line as:

```
iperf -s -w256k
```

The switch -s runs Iperf in server mode and -w is the window size being used, here 256 KBytes is being used. Iperf is started in client by the command line as:

```
iperf -c [Ip address of server] -w256k -i 2
```

Here the switch -c runs iperf in client mode and -i indicates the interval in terms of seconds between perodic bandwidth reports.

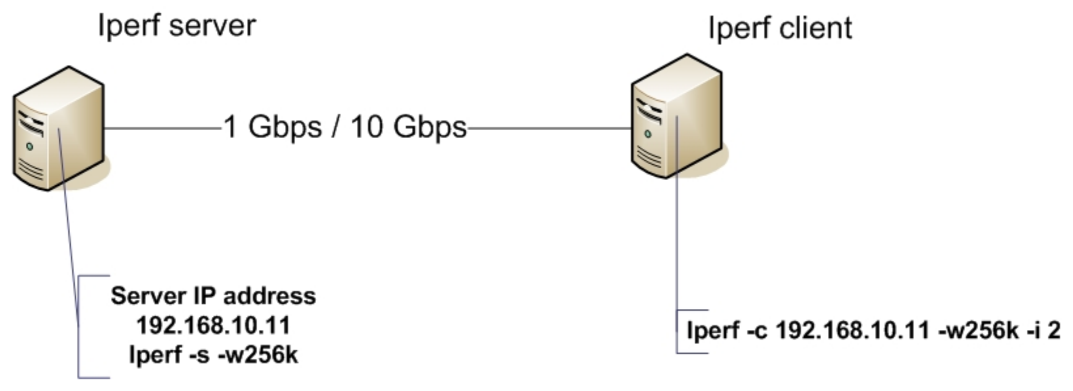


FIGURE 5.3: TCP Bandwidth measurement scenario with Iperf

Measuring the performance has been done on links of 1 Gbps and 10 Gbps between the systems.

Window size	**Link capacity**	**Delay**	**Throughput**
256 KB	1 Gbps	0.03 milli seconds	948 Mbps
256 KB	10 Gbps	0.03 milli seconds	9.88 Gbps
256 KB	10 Gbps	50 milli seconds	52.7 Mbps
256 KB	10 Gbps	100 milli seconds	26.4 Mbps
16 MB	10 Gbps	100 milli seconds	33.6 Mbps

TABLE 5.1: **Iperf TCP Performance**

As shown in table 5.1, a window size of 256 KB will give data transfer rate of 26.4 Mbps when delay is 100 ms. Now by increasing TCP window size to 16 MB the data rate is 33.6 Mbps. These measurements has been done with a packet loss of 0.1 percent.

5.4 Testing FTP using a direct TCP connection

First the data transfer rate for direct TCP connection of SmartFTP client with FTP-server is measured. Smart FTP-client is installed on a windows based machine with a dual core AMD Opteron(TM) processor, 2.60 GHz with 4 cores, 2.75 GBytes of RAM and a 1 Gbps network interface card. Vsftpd is used as FTP-server on Linux machine with dual core AMD opteron processor with 2.60 GHz with 4 cores and 7.85 GBytes of RAM. RAID is used as storage for high speed data transfer. The scenario is shown in figure 5.4. A throughput of 896 Mbps is observed with direct TCP connection on a 1 Gbps link by transferring a file of 8 GBytes.

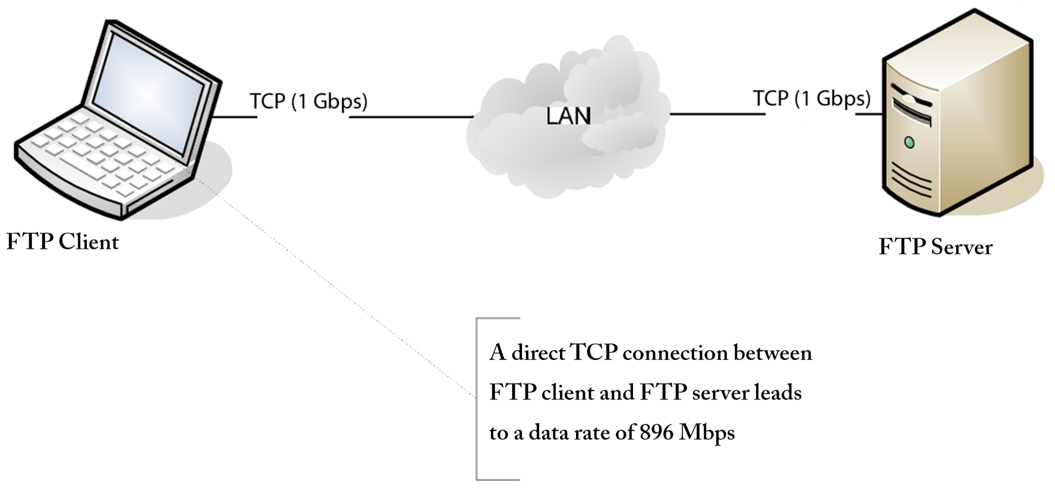

TCP (1 Gbps)
LAN
TCP (1 Gbps)
FTP Client
FTP Server
A direct TCP connection between FTP client and FTP server leads to a data rate of 896 Mbps

FIGURE 5.4: Testing Direct connection

5.5 Testing an indirect connection with TCP based-proxy

In this scenario the TCP-connection between FTP-client and server is now split into three connections. FTP-client is connected to a client-side proxy with a TCP connection, then client-side proxy makes a TCP connection to the server-side proxy and the third TCP connection is from server-side proxy to the FTP-server as shown in figure 5.5.

Each of the proxies is running on an Intel(R) Xeon(R) 3 GHz processor with 4 cores and 7.79 GBytes RAM.

When a file of 8 GBytes is transferred from FTP-server to FTP-client via server and client-side proxies, the throughput achieved is 808 Mbps.

The throughput is still not sufficient and not increased beyond 33.6 Mbps. This is because of the high RTT which will cause TCP degrading the throughput by many retransmissions due to time-out. That is why in case of data transfers over 10 Gbps link from client-side proxy to server-side proxy over the WAN enviroment in the presence of delay and packet losses a different protocol is used other than TCP so that the availability of 10 Gbps is properly utilized by moving to higher data rates for bulk data transfers in the presense of delay and packet losses.

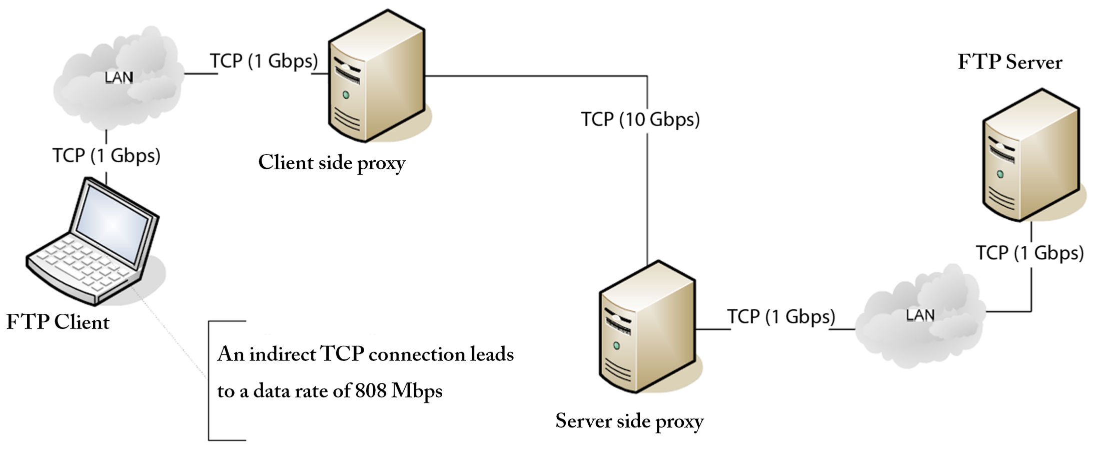


FIGURE 5.5: Testing an indirect connection with TCP based proxy

5.6 Testing an indirect connection with RWTP-based proxy

In this scenario there will be also an indirect connection from FTP-client to FTP-server using client-side and server-side proxies. However the proxies will use RWTP to make the connection between them as shown in figure 5.6.

The throughput achieved when transferring the same file of 8 GBytes from FTP-server to FTP-client via server-side and client-side proxies is 768 Mbps.

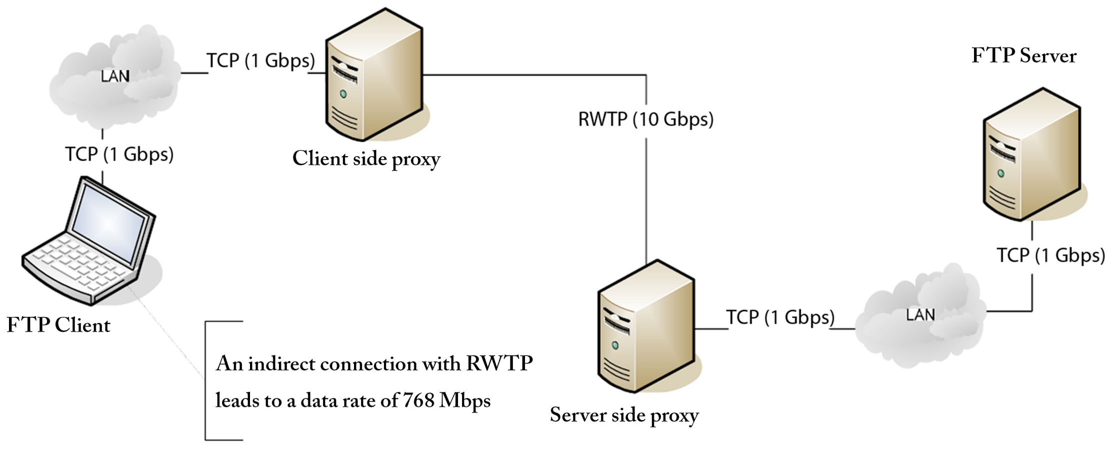


FIGURE 5.6: Testing an indirect connection with RWTP based proxy

5.7 Performance Evaluation

The performance evaluation is achieved by introducing a delay of 6 to 200 ms with two different packet loss rates of 0.1 percent and 0.3 percent by the aid of 10 Gbps network emulator. This is performed for TCP as well as RWTP in the scenario shown in figure 5.7.

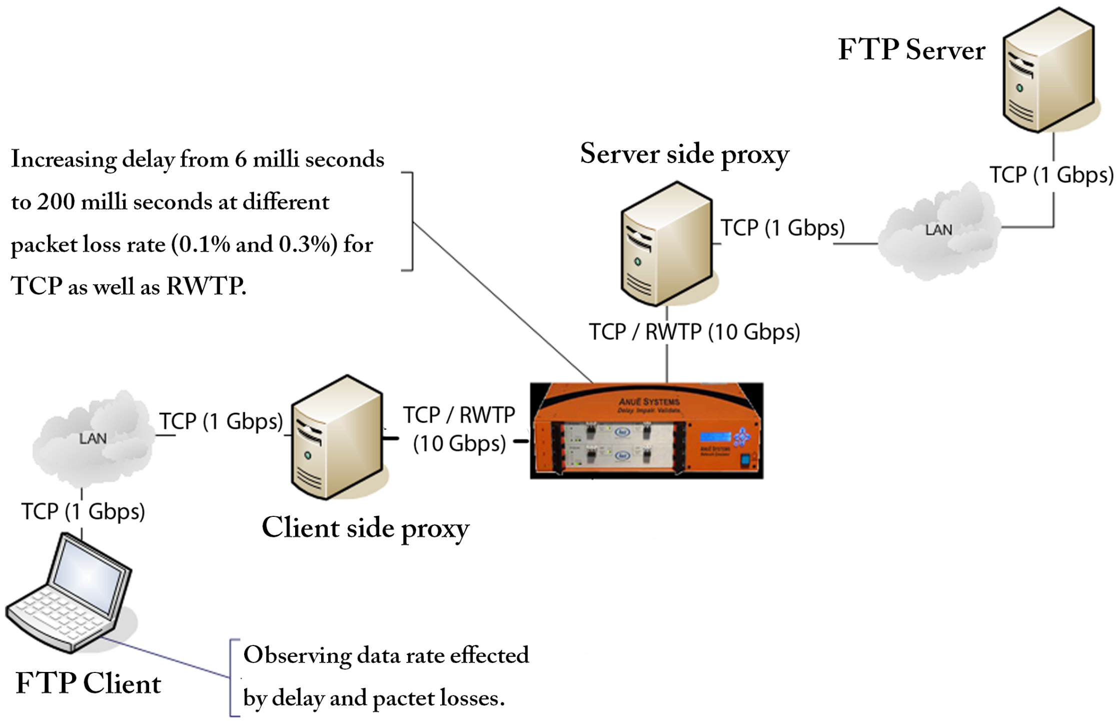

FTP Server
Server side proxy
TCP (1 Gbps)
LAN
TCP (1 Gbps)
Increasing delay from 6 milli seconds to 200 milli seconds at different packet loss rate (0.1% and 0.3%) for TCP as well as RWTP.
TCP / RWTP (10 Gbps)
LAN
TCP (1 Gbps)
TCP / RWTP (10 Gbps)
TCP (1 Gbps)
Client side proxy
FTP Client
Observing data rate effected by delay and pactet losses.

FIGURE 5.7: Performance analysis

The throughput measured at different delays with a packet loss of 0.1 percent is shown in figure 5.8.

RWTP performance is almost constant for delays between 12 and 50 ms. Having higher delay causes the performance of RWTP to degrade the throughput below 800 Mbps. But in case of TCP at 6 ms delay, the throughput is below 300 Mbps and it undergoes a continuous decrease as delay increases and becomes 14 Mbps at 200 ms.

Direct TCP connection, from FTP-client to FTP-server is slightly worse than TCP connection using a proxy. This is so, because when using proxies each TCP connection can be handled with a shorter response time as compared to larger response time when direct TCP connection is used across bigger networks [16]. This is especially important when congestion control mechanism is considered as it has strong impact on TCP performance.

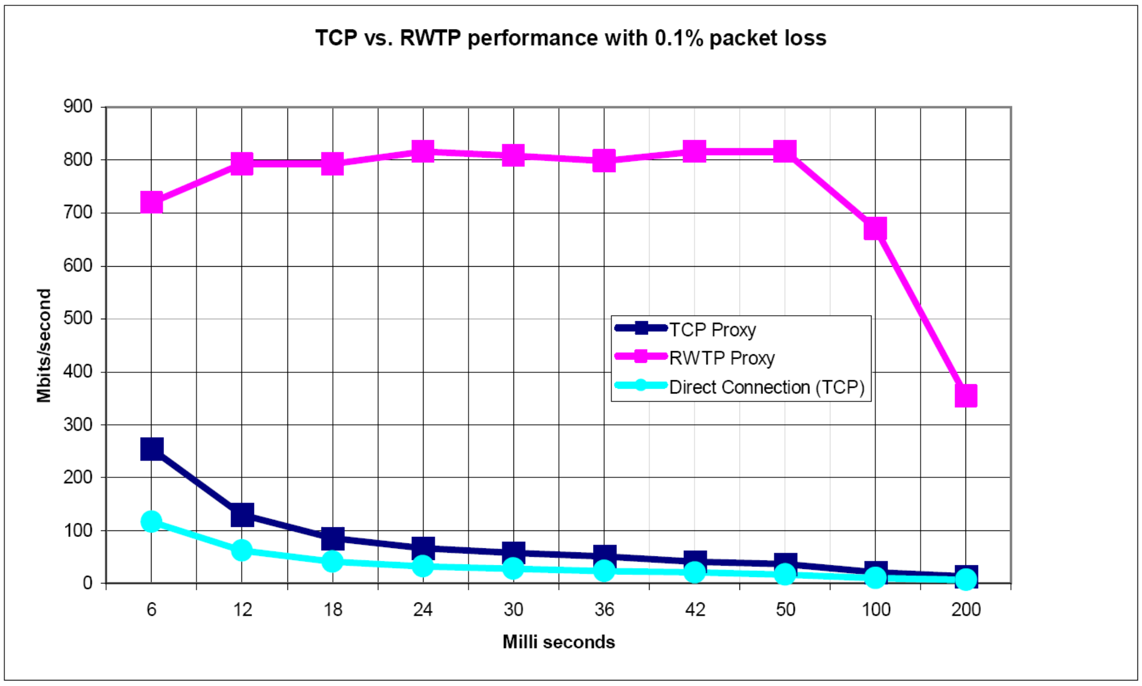


FIGURE 5.8: TCP vs RWTP with 0.1 percent packet loss

Considering the data rate at 50 ms and 0.1 percent packet loss RWTP gives 816 Mbps while a direct TCP connection gives 17 Mbps, hence giving a brilliant factor of 48. Similarly, TCP with proxy gives 36 Mbps which makes the factor equal to 22.

Now considering the data rate at 50 ms and 0.3 percent packet loss RWTP gives 808 Mbps and a direct TCP connection gives 15 Mbps, hence giving a factor of 53. Similarly, TCP with proxy gives 20 Mbps which makes the factor equal to 40.the performance of TCP vs. RWTP in the presence of delay is shown in figure 5.9.

From the plot it is shown that RWTP almost remains at a constant rate below a delay of 50 ms, as soon as delay increases more than 50 ms a degradation in performance is shown in figure 5.9. This is because of the limitations of buffer sizes and calculated delay (distance of 10,000 kilometers).

With a packet loss of 0.3 percent the data rate at 6 ms delay is below 200 Mbps and continues to decrease as delay increases and becomes 6 Mbps as delay reaches 200 ms. Similarly, performance of direct connection using TCP is slightly worse than TCP with proxy. However, comparing RWTP with TCP there is still a huge gap in performance between them.

RWTP gives a much better data transfer rate as compared to direct FTP connections or proxy based using TCP. Transferring a file of 125 GBytes on a network with a distance

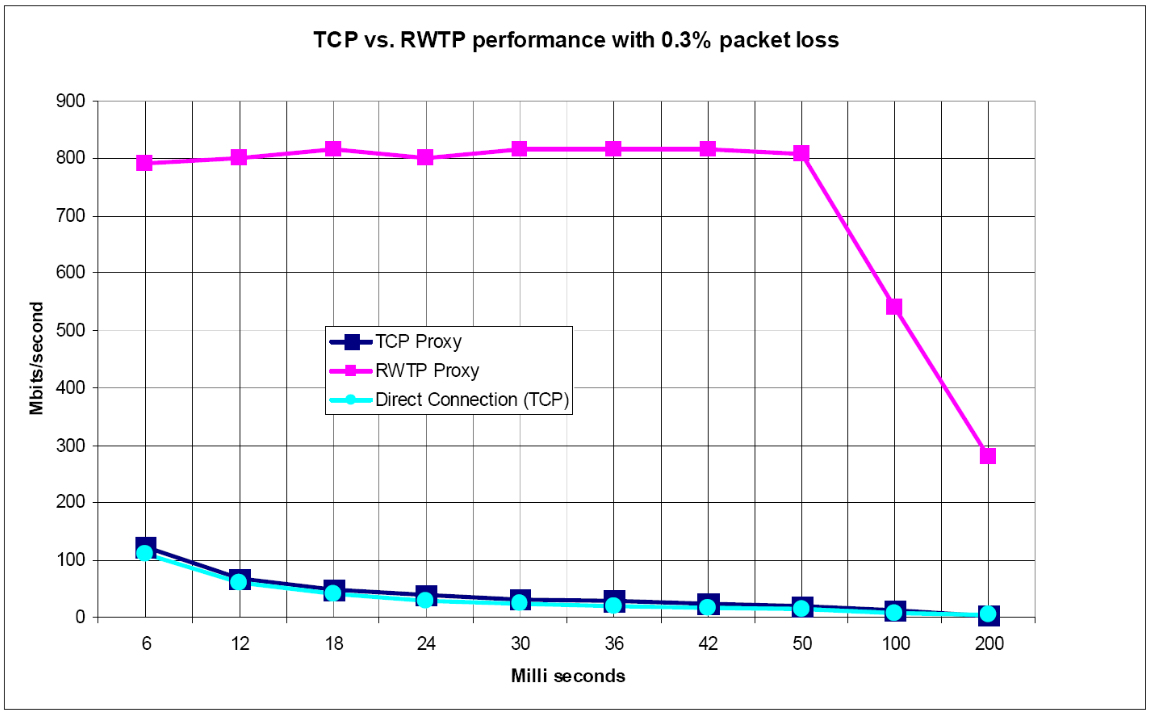


FIGURE 5.9: TCP vs RWTP with 0.3 percent packet loss

of 10,000 kilometers (delay of approximately 50 ms) from Hannover, Germany to Los Angeles, United States with 0.3 percent packet loss the results are shown in table 5.2.

Connection type	Data rate	Transfer Time
Direct	15 Mbps	19 hours
TCP Proxy	20 Mbps	14 hours
RWTP Proxy	808 Mbps	21 minutes

TABLE 5.2: **Performance analysis table for TCP-proxy, RWTP-proxy and direct connection**

Using 4K digital technology with 5000 frames each frame contains 50 MBytes which makes a total of 250 GBytes can be transfered from Hannover, Germany to Los Angeles, United States in 42 minutes using RWTP. On the other hand with a direct FTP connection using TCP it takes 37 hours but only 28 hours with a proxy-based FTP connection using TCP.

Since, RWTP performs better in terms of high-speed data transfer, client-side and server-side proxy using RWTP protocol on data channel has, as expected, shown a better performance.

Hence, RWTP-Gateway with FTP is an appropriate solution for transferring bulk data over high speed networks in the presence of delay and packet losses.

Chapter 6

Summary and Future work

In this thesis work the protocol RWTP is examined with a proxy mechanism collaborating with FTP on 1 Gbps Ethernet networks interconnected by 10 Gbps routers and the network model performance is investigated using 10 Gbps network emulator. To make RWTP perform collaborative tasks with FTP a proxy mechanism has been designed so that a proxy gateway is implemented for each end between FTP-client and server. Each proxy is capable of converting transport protocol from TCP to RWTP so that RWTP is used over WAN and TCP is used locally. Control channel uses TCP throughout, on the other hand data channel uses TCP from FTP-client to client-side proxy. Similarly, from server-side proxy to FTP server TCP is used. In order to enhance performance capability the send and receive buffer size is tuned by extending the default values. After implementation several tests were run to evaluate the implementation. The results presented in chapter 5 shows that by adopting a proxy mechanism for high data rate using RWTP, the collaborative tasks can be performed with common existing applications like FTP so that high data rate capability is properly utilized.

During the work some ideas arised regarding the further development of RWTP proxy. One idea is to implement this proxy scenario providing support for HTTP, SAMBA, NFS and other most common file sharing protocols.

Enhancement in the proxies can also be made by implementing caching mechanisms. For example client side can store data responses for future requests. On subsequent requests the cache can help serving and responding without consuming any additional resources. This caching mechanism has been shown to be fruitful for applications like HTTP. In order that the proxy will work well with FTP caching it needs to have file storing capabilities. This task should be investigated more in the future.

Appendix A

Network performance tables

Connection type	Data rate
Direct	896 Mbps
TCP Proxy	808 Mbps
RWTP Proxy	768 Mbps

Table A.1: **Performance analysis table for TCP Proxy, RWTP Proxy and direct connection without emulator**

Connection type	**Delay**	**Data rate**
Direct	6 milli seconds	117 Mbps
TCP Proxy	6 milli seconds	254 Mbps
RWTP Proxy	6 milli seconds	720 Mbps
Direct	12 milli seconds	62 Mbps
TCP Proxy	12 milli seconds	130 Mbps
RWTP Proxy	12 milli seconds	793 Mbps
Direct	18 milli seconds	42 Mbps
TCP Proxy	18 milli seconds	85 Mbps
RWTP Proxy	18 milli seconds	653 Mbps
Direct	24 milli seconds	32 Mbps
TCP Proxy	24 milli seconds	67 Mbps
RWTP Proxy	24 milli seconds	816 Mbps
Direct	30 milli seconds	28 Mbps
TCP Proxy	30 milli seconds	58 Mbps
RWTP Proxy	30 milli seconds	808 Mbps
Direct	36 milli seconds	24 Mbps
TCP Proxy	36 milli seconds	51 Mbps
RWTP Proxy	36 milli seconds	798 Mbps
Direct	42 milli seconds	21 Mbps
TCP Proxy	42 milli seconds	42 Mbps
RWTP Proxy	42 milli seconds	808 Mbps
Direct	50 milli seconds	17 Mbps
TCP Proxy	50 milli seconds	36 Mbps
RWTP Proxy	50 milli seconds	816 Mbps
Direct	100 milli seconds	10 Mbps
TCP Proxy	100 milli seconds	21 Mbps
RWTP Proxy	100 milli seconds	671 Mbps
Direct	200 milli seconds	7 Mbps
TCP Proxy	200 milli seconds	14 Mbps
RWTP Proxy	200 milli seconds	354 Mbps

TABLE A.2: **Performance analysis table for TCP Proxy, RWTP Proxy and direct connection with 0.1 percent packet loss**

Connection type	**Delay**	**Data rate**
Direct	6 milli seconds	122 Mbps
TCP Proxy	6 milli seconds	122 Mbps
RWTP Proxy	6 milli seconds	792 Mbps
Direct	12 milli seconds	60 Mbps
TCP Proxy	12 milli seconds	68 Mbps
RWTP Proxy	12 milli seconds	800 Mbps
Direct	18 milli seconds	40 Mbps
TCP Proxy	18 milli seconds	48 Mbps
RWTP Proxy	18 milli seconds	816 Mbps
Direct	24 milli seconds	30 Mbps
TCP Proxy	24 milli seconds	32 Mbps
RWTP Proxy	24 milli seconds	800 Mbps
Direct	30 milli seconds	25 Mbps
TCP Proxy	30 milli seconds	32 Mbps
RWTP Proxy	30 milli seconds	816 Mbps
Direct	36 milli seconds	20 Mbps
TCP Proxy	36 milli seconds	28 Mbps
RWTP Proxy	36 milli seconds	816 Mbps
Direct	42 milli seconds	17 Mbps
TCP Proxy	42 milli seconds	24 Mbps
RWTP Proxy	42 milli seconds	816 Mbps
Direct	50 milli seconds	15 Mbps
TCP Proxy	50 milli seconds	20 Mbps
RWTP Proxy	50 milli seconds	808 Mbps
Direct	100 milli seconds	8 Mbps
TCP Proxy	100 milli seconds	11 Mbps
RWTP Proxy	100 milli seconds	541 Mbps
Direct	200 milli seconds	4 Mbps
TCP Proxy	200 milli seconds	6 Mbps
RWTP Proxy	200 milli seconds	280 Mbps

TABLE A.3: **Performance analysis table for TCP Proxy, RWTP Proxy and direct connection with 0.3 percent packet loss**

Appendix B

CPU usage plots

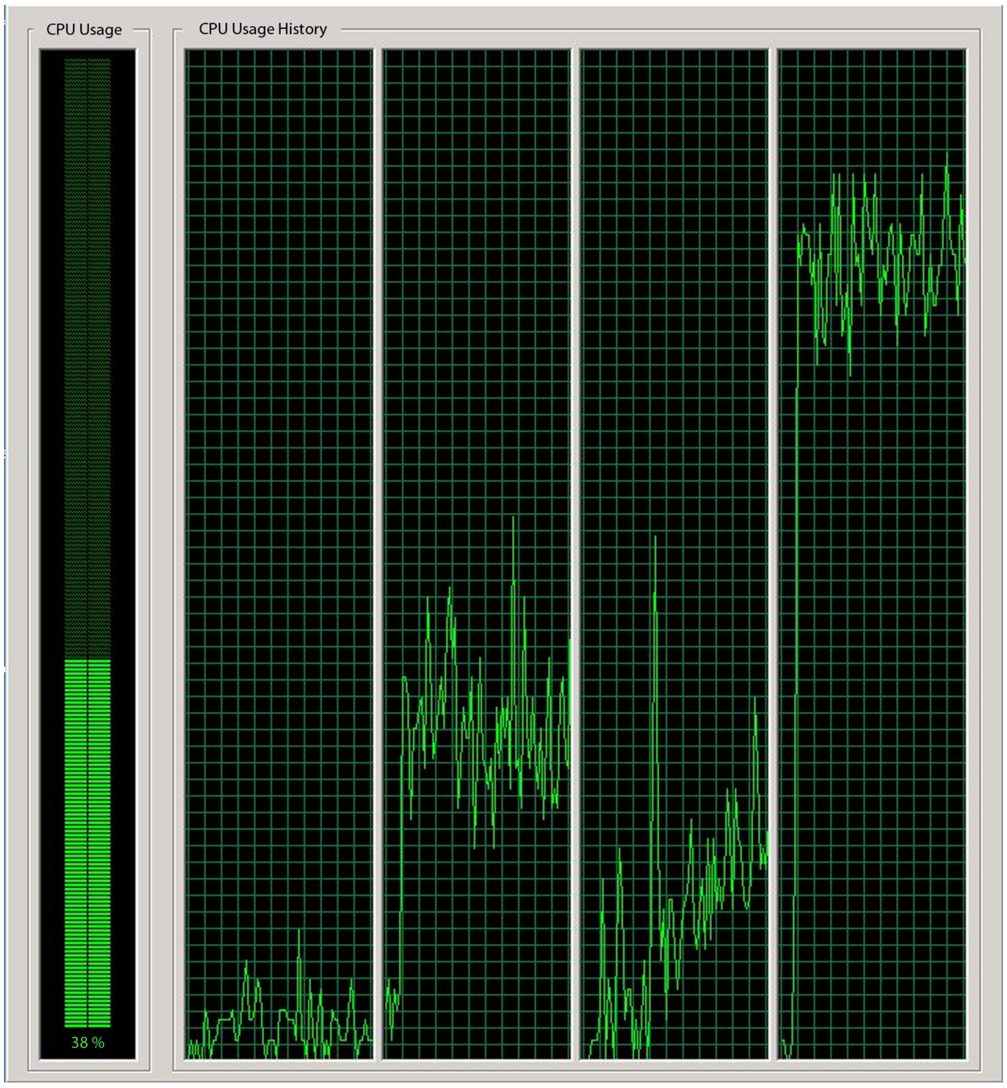
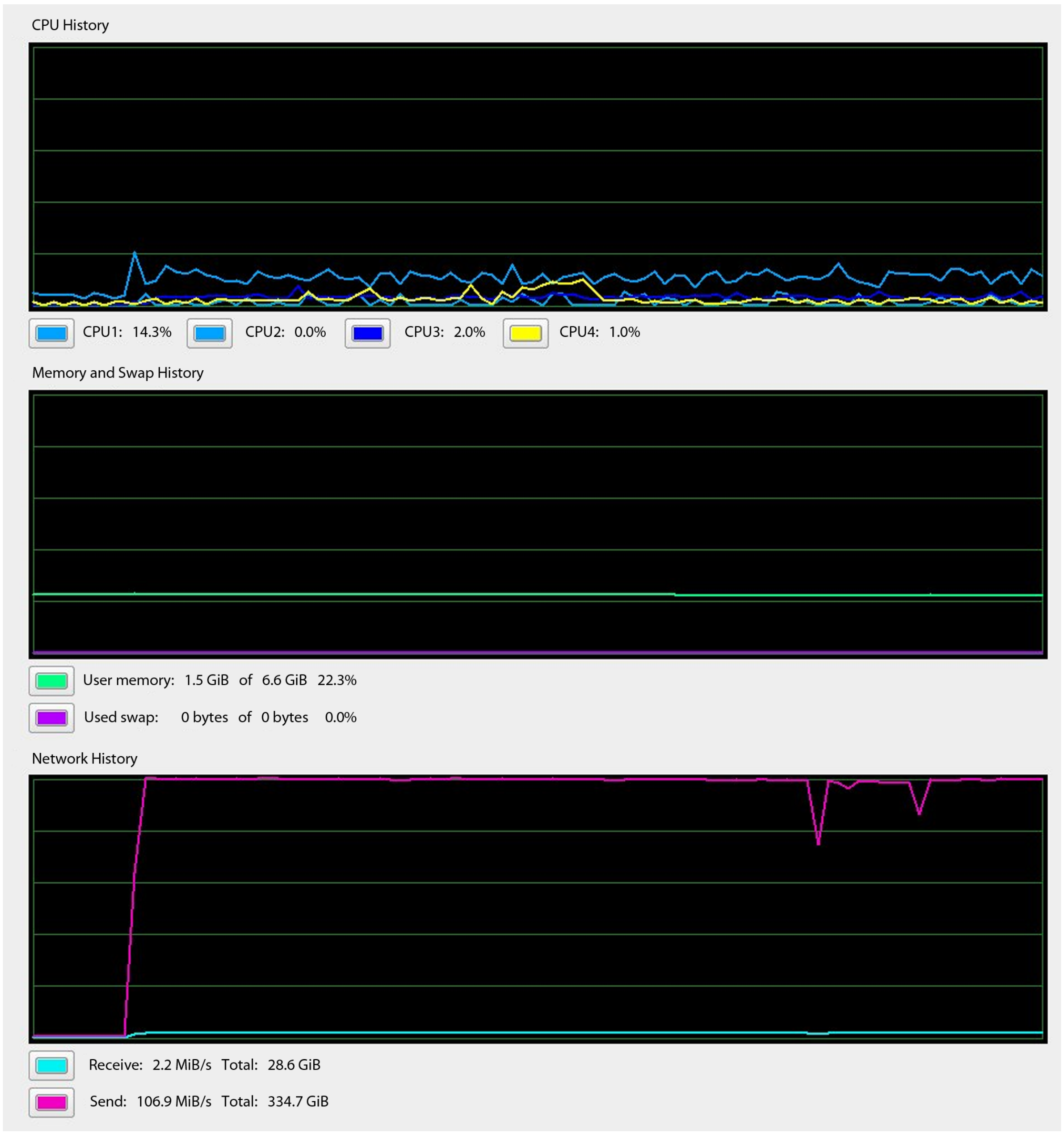


FIGURE B.1: CPU Usage at FTP client



🟦 CPU1: 14.3% 🟦 CPU2: 0.0% 🟦 CPU3: 2.0% 🟨 CPU4: 1.0%

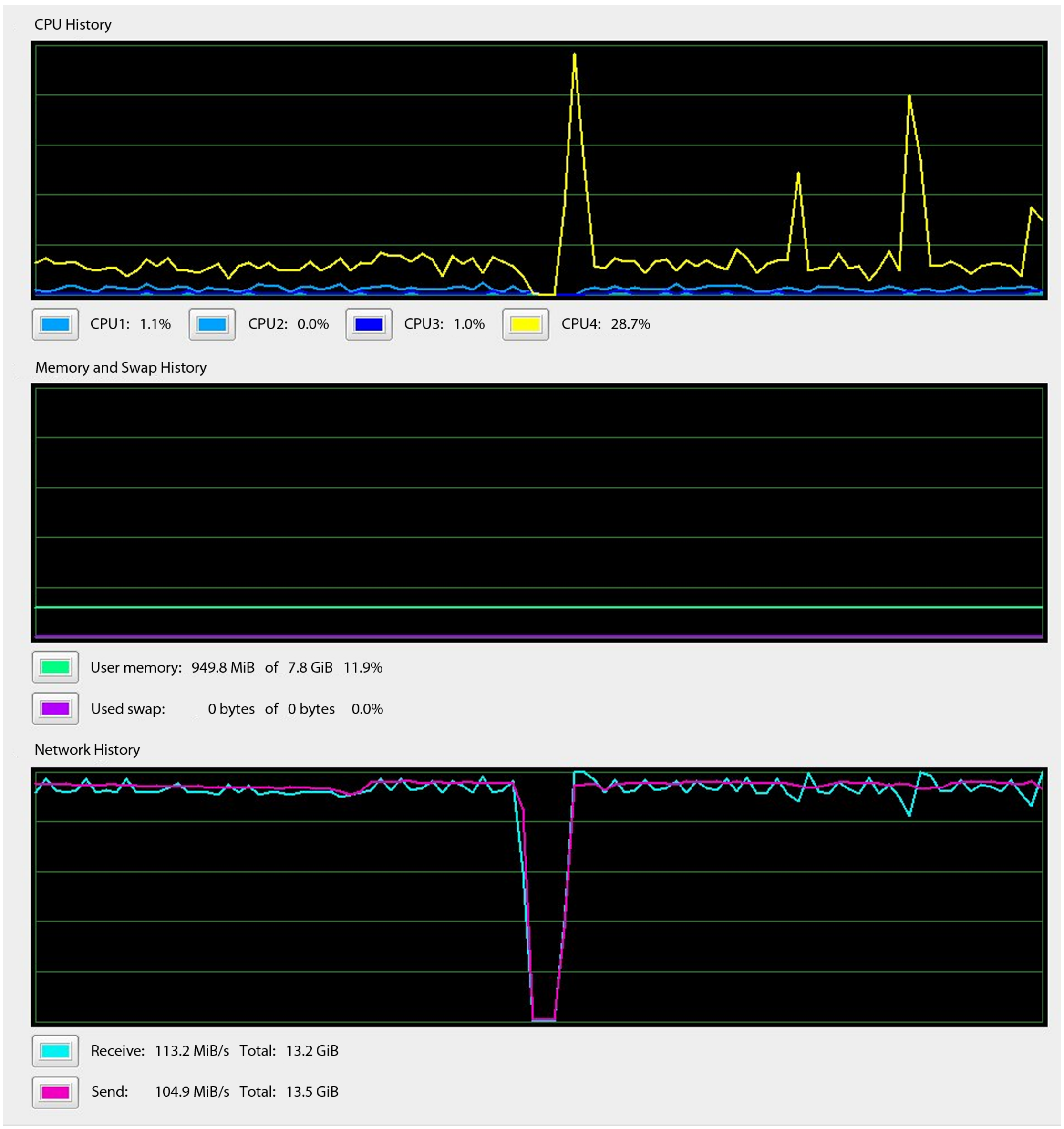
🟩 User memory: 1.5 GiB of 6.6 GiB 22.3%

🟪 Used swap: 0 bytes of 0 bytes 0.0%

🟦 Receive: 2.2 MiB/s Total: 28.6 GiB

🟪 Send: 106.9 MiB/s Total: 334.7 GiB

FIGURE B.2: CPU Usage at FTP server.


CPU History

🟦 CPU1: 1.1% 🟦 CPU2: 0.0% 🟦 CPU3: 1.0% 🟨 CPU4: 28.7%

Memory and Swap History

🟩 User memory: 949.8 MiB of 7.8 GiB 11.9%

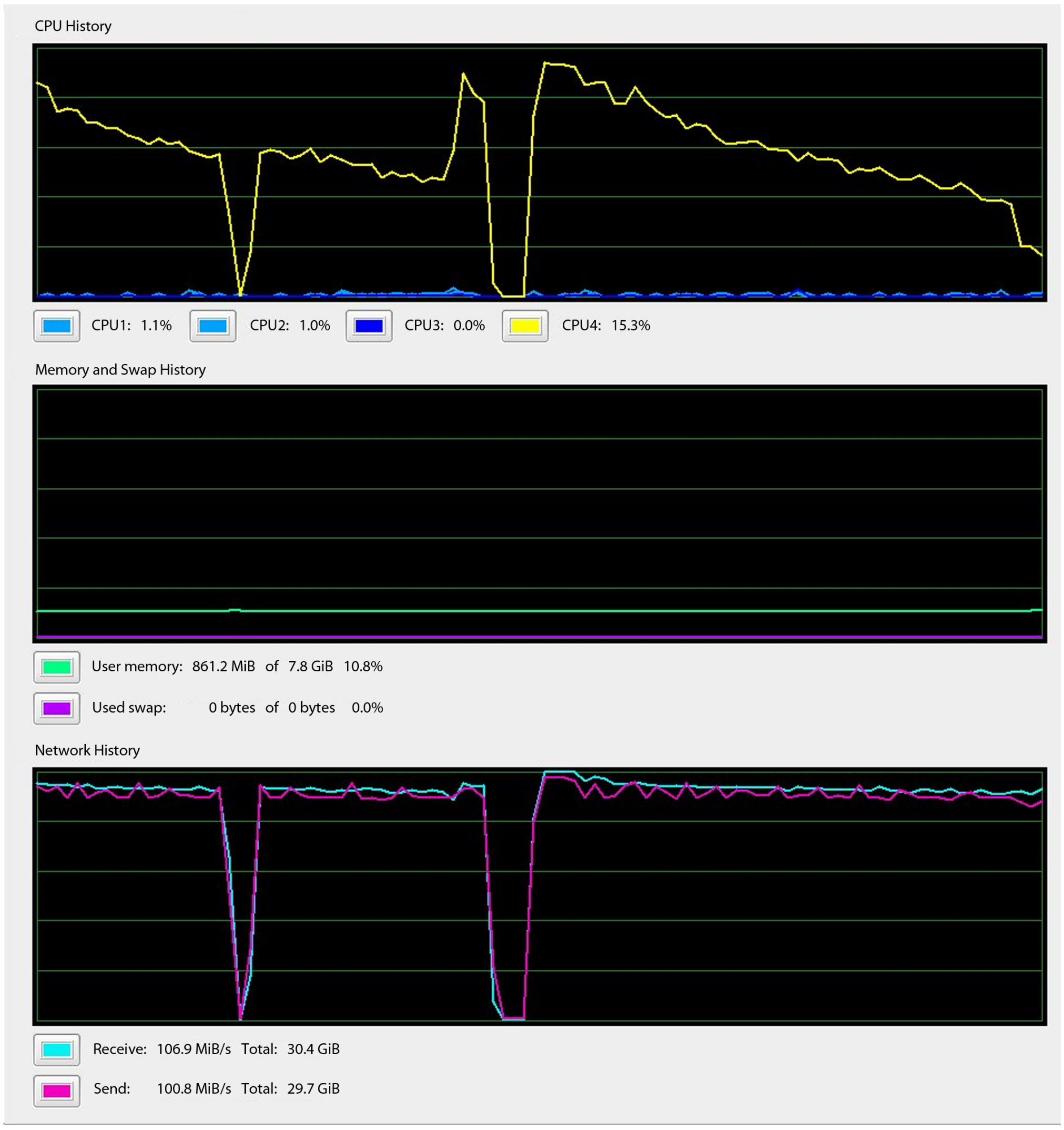
🟪 Used swap: 0 bytes of 0 bytes 0.0%

Network History

🟦 Receive: 113.2 MiB/s Total: 13.2 GiB

🟪 Send: 104.9 MiB/s Total: 13.5 GiB

FIGURE B.3: CPU Usage at Client side proxy (TCP as WAN connection). Sudden fall or rise in network plot and CPU usage plot identifies the completion of one data transfer or start of another data transfer.



🟦 CPU1: 1.1% 🟦 CPU2: 1.0% 🟦 CPU3: 0.0% 🟨 CPU4: 15.3%

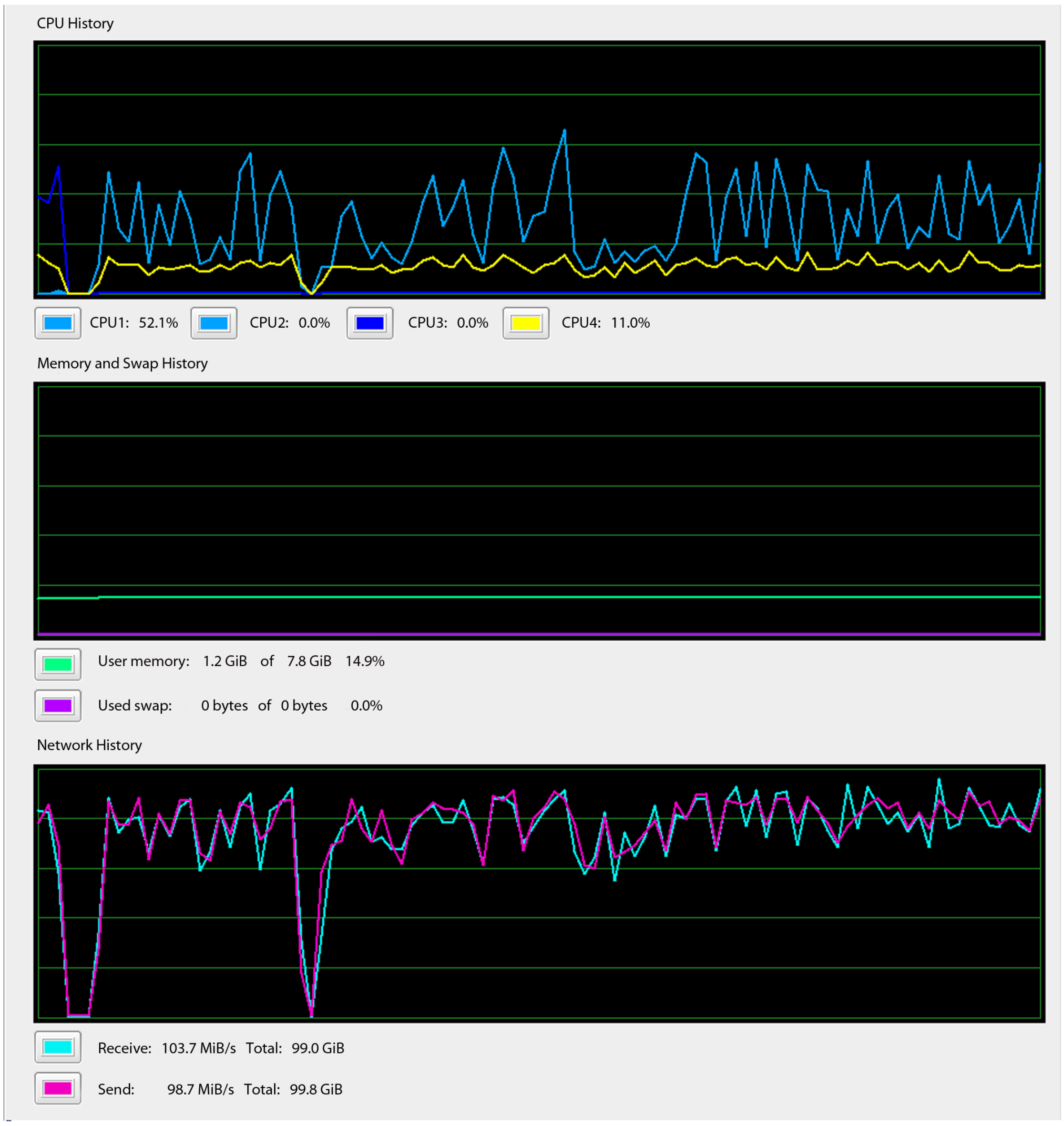
🟩 User memory: 861.2 MiB of 7.8 GiB 10.8%

🟪 Used swap: 0 bytes of 0 bytes 0.0%

🟦 Receive: 106.9 MiB/s Total: 30.4 GiB

🟪 Send: 100.8 MiB/s Total: 29.7 GiB

Figure B.4: CPU Usage at Server side proxy (TCP as WAN connection). Sudden fall or rise in network plot and CPU usage plot identifies the completion of one data transfer or start of another data transfer.



CPU1: 52.1% CPU2: 0.0% CPU3: 0.0% CPU4: 11.0%

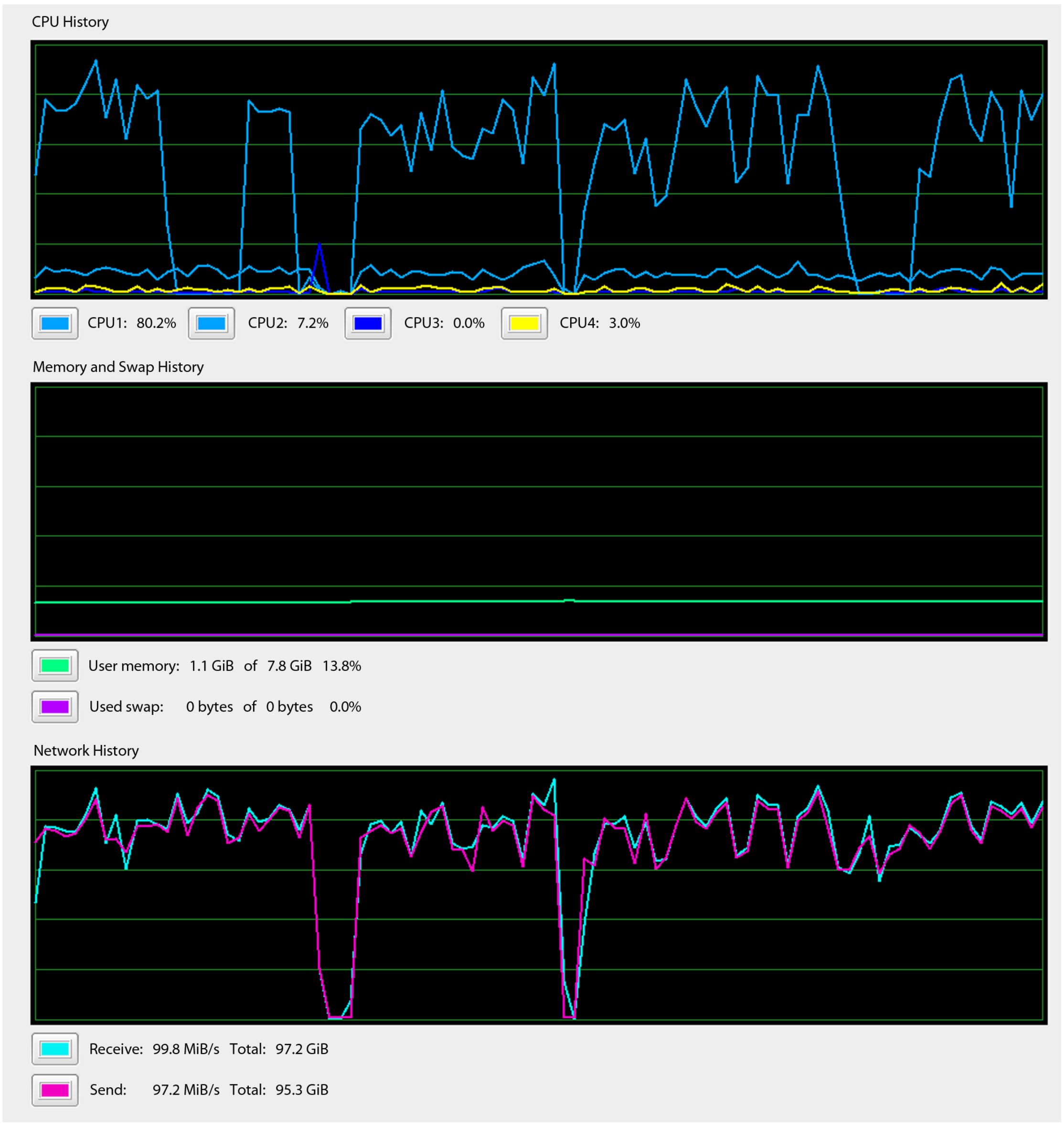
User memory: 1.2 GiB of 7.8 GiB 14.9%

Used swap: 0 bytes of 0 bytes 0.0%

Receive: 103.7 MiB/s Total: 99.0 GiB

Send: 98.7 MiB/s Total: 99.8 GiB

Figure B.5: CPU Usage at Client side proxy (RWTP as WAN connection). Sudden fall or rise in network plot and CPU usage plot identifies the completion of one data transfer or start of another data transfer.



CPU1: 80.2% CPU2: 7.2% CPU3: 0.0% CPU4: 3.0%

User memory: 1.1 GiB of 7.8 GiB 13.8%

Used swap: 0 bytes of 0 bytes 0.0%

Receive: 99.8 MiB/s Total: 97.2 GiB

Send: 97.2 MiB/s Total: 95.3 GiB

FIGURE B.6: CPU Usage at Server side proxy (RWTP as WAN connection). Sudden fall or rise in network plot and CPU usage plot identifies the completion of one data transfer or start of another data transfer.

Bibliography

[1] Miniwatts Marketing Group. Internet usage statistics, the internet big picture. *Internet World Stats*, June 2009. URL `http://www.internetworldstats.com/stats.htm`.

[2] Charles Kozierok. The tcp/ip guide: A comprehensive, illustrated internet protocols reference. page 1648, October 2005.

[3] Kiran Sultan Habibullah Jamal. Performance analysis of tcp congestion control algorithms. *International Journal of Computers and communications*, 2:9, 2008. URL `http://www.wseas.us/journals/cc/cc-27.pdf`.

[4] Stuart Cheshire. It's the latency, stupid. May 1996. URL `http://www.stuartcheshire.org/rants/Latency.html`.

[5] Sony. Fact book 2006. December 2006. URL `http://www.sony.net/SonyInfo/IR/library/fact/FY06_3Q.pdf`.

[6] Don Johnson. Communication protocols. URL `http://cnx.org/content/m0080/latest/`.

[7] John Daintith. Packet. *A Dictionary of Computing*, 2004. URL `http://www.encyclopedia.com/doc/1O11-packet.html`.

[8] Bradley Mitchell. Osi model reference guide. URL `http://compnetworking.about.com/cs/designosimodel/a/osimodel.htm`.

[9] Alan Fekete Nancy Lynch, Yishay Mansour. The data link layer: Two impossibility results. *Laboratory for Computer science, Massachusetts Institute of Technology*, 1988. URL `http://groups.csail.mit.edu/tds/papers/Lynch/podc88-datalink.pdf`.

[10] J. (ed.) Postel. Internet protocol - darpa internet program protocol specification. *USC/Information Sciences Institute*, September 1981.

[11] California 90291 Information Sciences Institute University of Southern California 4676 Admiralty Way Marina del Rey. Transmission control protocol. *DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*, September 1981.

[12] Jitendra Padhye Sally Floyd, Mark Handley. A comparison of equation-based and aimd congestion control. *ACIRI*, page 12, May 2000. URL `http://www.icir.org/tfrc/aimd.pdf`.

[13] Steve Thompson. The cold hard truth about tcp/ip performance over the wan. *Computer Technology Review*, August 2004. URL `http://findarticles.com/p/articles/mi_m0BRZ/is_8_24/ai_n7072133/?tag=content;col1`.

[14] M. Hassan and R. Jain. High performance tcp/ip networking: Concepts, issues, and solutions. 1, 2003.

[15] J. Crowcroft B. Davie S. Deering D. Estrin S. Floyd V. Jacobson G. Minshall C. Partridge L. Peterson K. Ramakrishnan S. Shenker J. Wroclawski L. Zhang B. Braden, D. Clark. Recommendations on queue management and congestion avoidance in the internet. *Network Working Group*, 1, April 1998. URL `http://www.faqs.org/rfcs/rfc2309.html`.

[16] J. Griner G. Montenegro Z. Shelby J. Border, M. Kojo. Performance enhancing proxies intended to mitigate link-related degradations. *Network Working Group*, June 2001. URL `http://www.isi.edu/in-notes/rfc3135.txt`.

[17] J. Postel. Rfc 768 user datagram protocol. *Network Working Group*, August 1980. URL `http://www.ietf.org/rfc/rfc0768.txt`.

[18] J. Postel. Rfc 959 - file transfer protocol. *Network Working Group*, October 1985. URL `http://www.faqs.org/rfcs/rfc959.html`.

[19] D. Brent Chapman Elizabeth D. Zwicky, Simon Cooper. Building internet firewalls. 2, June 2000.

[20] Mike Gleason. The file transfer protocol (ftp) and your firewall / network address translation (nat) router / load-balancing router. *NcFTP Software*, April 2005. URL `http://www.ncftp.com/ncftpd/doc/misc/ftp_and_firewalls.html`.

[21] Andrew R. Hickey. The wan speeds up. *TechTarget ANZ*, November 2005. URL `http://searchnetworking.techtarget.com.au/articles/17491-The-WAN-speeds-up`.

[22] 100 gigabit ethernet. . URL `http://en.wikipedia.org/wiki/40_Gigabit_Ethernet`.

[23] University of Illinois at Chicago. Udt: Udp based data transfer protocol. breaking the data transfer bottleneck. *National Center for Data Mining.* URL `http://udt.sourceforge.net/udt-sc08-poster.pdf`.

[24] Robert L. Grossman Yunhong Gu. Udt: Udp-based data transfer for high-speed wide area networks. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 51:1777–1799, May 2007. URL `http://portal.acm.org/citation.cfm?id=1229240`.

[25] Talon Data Systems. Netflight. URL `http://www.talondata.com/pdf/servers/network/net_flight/NetFlightDatasheet.pdf`.

[26] Bruce S. Davie Larry L. Peterson. Computer networks: a systems approach. 2007.

[27] Proxy server. *whatis?com*, December 2008. URL `http://whatis.techtarget.com/definition/0,,sid9_gci212840,00.html`.

[28] J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee R. Fielding, J. Gettys. Hypertext transfer protocol – http/1.1. June 1999. URL `http://tools.ietf.org/html/rfc2616#page-46`.

[29] M. Chatel. Classical versus transparent ip proxies. *Network Working Group*, March 1996. URL `http://www.ietf.org/rfc/rfc1919.txt`.

[30] Scott Mueller Terry William Ogletree. Upgrading and repairing networks. page 807, 2006.

[31] Min Sik Kim Xincheng Zhang Simon S. Lam Y. Richard Yang, Yang Richard Yang. Two problems of tcp aimd congestion control. June 2000. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.3446`.

[32] D. Borman V. Jacobson, R. Braden. Tcp extensions for high performance. *Network Working Group*, May 1992. URL `http://www.ietf.org/rfc/rfc1323.txt`.

[33] Network emulation. . URL `http://en.wikipedia.org/wiki/Network_emulation`.

[34] Xgem 10g ethernet network emulators. . URL `http://www.anuesystems.com/`.

[35] Tcp tuning guide. 2009. URL `http://fasterdata.es.net/TCP-tuning/`.

[36] Oskar Andreasson. Ipsysctl tutorial 1.0.4. 2002. URL `http://ipsysctl-tutorial.frozentux.net/chunkyhtml/tcpvariables.html`.