HIGH THROUGHPUT UDP-BASED PEER-TO-PEER SECURE DATA TRANSFER

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Fadime Tuğba Doğan
May 2018

High Throughput UDP-based Peer-to-Peer Secure Data Transfer
By Fadime Tuğba Doğan
May 2018

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Can Alkan(Advisor)

Çiğdem Gündüz Demir

Aybar Can Acar

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

HIGH THROUGHPUT UDP-BASED PEER-TO-PEER SECURE DATA TRANSFER

Fadime Tuğba Doğan
M.S. in Computer Engineering
Advisor: Can Alkan
May 2018

High throughput sequencing (HTS) platforms have been developed in recent years. These technologies enable researchers to answer a wide range of biological questions by obtaining whole or targeted segments of genomes of individuals. However, HTS technologies generate very large amounts of data. Even after using the best compression algorithms, data size is still huge due to large original file size. As most of the genome projects' contributors are located in different countries, transfer of the data becomes an important problem in genomics. Currently used methods for genome data sharing is transferring the files via File Transfer Protocol (FTP), Tsunami protocol or Aspera Software, storing them on public databases or clouds, working on the files stored on central servers and circulating external hard disks. However, all of these methods have some drawbacks like cost, speed, or privacy. In this thesis, to address this problem, we introduce an application called BioPeer. BioPeer uses an open source UDP-based UDT protocol written by Barchart, Inc for data transfer. We implement peer-to-peer file sharing architecture to BioPeer. This architecture is similar to BitTorrent, where large files are transferred in chunks, and synchronized between peers within the same project. To ensure every client is able to connect other clients, we employ NAT traversal via UDP hole punching method. So, users who are behind NAT devices are able to send and receive data from other peers. To provide secure file transfer, BioPeer encrypts files using Advanced Encryption Standard (AES) cipher. Symmetric encryption keys are exchanged via RSA (Rivest-Shamir-Adleman) algorithm. Additionally, content distribution network (CDN) infrastructure is implemented in order to achieve high throughput with BioPeer.

Keywords: UDP, data transfer, peer-to-peer, big data.

ÖZET

YÜKSEK ÇIKTILI, KULLANICI VERİBLOĞU İLETİŞİM KURALLARI (UDP) TABANLI, EŞLER ARASI, GÜVENLİ VERİ AKTARIMI

Fadime Tuğba Doğan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Can Alkan
Mayıs 2018

Son yıllarda yüksek verimli sıralama (HTS) platformları geliştirilmiştir. Araştırmacılar bu teknolojileri kullanarak bireylerin genomlarının tamamı veya hedeflenmiş kısımlarını elde edebilir. Elde edilen bu veriler araştırmacıların çeşitli biyolojik sorulara cevap vermelerini sağlar. Ancak yüksek verimli sıralama teknolojilerindeki gelişmeler çok büyük miktarda veri üretilmesine neden olur. Dosyaların asıl boyutları çok büyük olduğu için en iyi sıkıştırma algoritmalarının uygulanması bile veri miktarını çok fazla azaltamaz. Genom projelerinde yer alan katılımcılar genellikle farklı ülkelerde yer almaktadır. Bu yüzden genom projelerindeki büyük dosyaların aktarımı sorun haline gelmektedir. Araştırmacıların şu anda kullandıkları yöntemler şunlardır; dosya aktarım iletişim kuralı (FTP), Tsunami iletişim kuralı veya Aspera yazılımı ile veri aktarımı, dosyaları herkese açık veritabanlarında yada bulutta depolamak, dosyaları ortak sunucuda tutup onun üzerinde çalışmak ya da verileri harici disklere kaydedip katılımcılar arasında dolaştırmak. Ancak tüm bu yöntemlerin maliyet, hız ve gizlilik gibi dezavantajları vardır. Bu sorunları ortadan kaldırmak için BioPeer adında bir uygulama geliştirdik. Bu uygulama, veri aktarımı için Barchart şirketi tarafından yazılmış açık kaynaklı UDP tabanlı veri aktarım protokolü UDT kullanıyor. BioPeer'e eşler arası dosya paylaşım (P2P) mimarisi uygulanmıştır. Bu mimari, BitTorrent'te olduğu gibi büyük dosyaları küçük parçalara ayırarak aynı projede yer alan eşler arasında senkronize eder. Tüm kullanıcıların birbirine bağlanabildiğinden emin olmak için NAT'ı Aşma (NAT Traversal) yöntemleri arasından UDP Delik Açma (UDP Hole Punching) yöntemi kullanılmıştır. Bu sayede NAT cihazları arkasında kalan kullanıcılar da diğer katılımcılarla dosya alışverişi yapabilir. Uygulamada güvenli veri aktarımı sağlamak için, dosyalar Gelişmiş

Şifreleme Standardı (AES) kullanılarak şifrelenir. Simetrik şifreleme anahtarları RSA (Rivest-Shamir-Adleman) algoritması kullanılarak kullanıcılar arasında aktarılır. Ayrıca, uygulamadan daha yüksek verim elde etmek için, içerik dağıtım ağı (CDN) altyapısı uygulanmıştır.

Anahtar sözcükler: UDP, veri transferi, eşler arası (P2P), büyük veri.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my advisor Assist. Prof. Can Alkan for the continuous support of my research, for his guidance, suggestions and patience throughout my studies.

I would also like thank to the love of my life, my husband, Cihad Öge, for being in my life, his endless love, patience, faith, and help. I couldn't finish my research and thesis without his support and help.

Last but not least, I would like to thank my family; my father Türk Aslan, my mother Türkan and my brother Alp Eren for their endless and unconditional love, encouragement, and support in every period of my life. I wouldn't be who I am without them and my husband. I am grateful to you.

Contents

1 Introduction **1**

1.1 Current Methods for Genome Data Sharing 4

1.2 Contribution of the Thesis 8

1.3 Organization of the Thesis 8

2 BioPeer: High Throughput UDP-based Peer-to-Peer Secure Data Transfer **9**

2.1 Overview of BioPeer 9

2.2 Features of BioPeer 10

2.2.1 File Transfer Using UDP-based Data Transfer (UDT) Protocol 10

2.2.2 Secure File Transfer 13

2.2.3 Peer-to-Peer (P2P) File Sharing 14

2.2.4 NAT-Traversal 17

2.3 Content Distribution Network (CDN) 22

2.3.1 CDN Implementation of BioPeer 23

2.4 Sharing Data in Public Projects 25

3 Performance Evaluation of BioPeer 26

4 Conclusion & Future Work 33

4.1 Conclusion 33

4.2 Future Work 34

List of Figures

1.1 RTT comparison for FTP-UDT protocols in 1 Gbps link with 10GB file transfer. Adapted from [14]. 5

1.2 Performance Evaluation of FTP vs. UDT protocols with different file sizes. Adapted from [14]. 6

2.1 Comparison between UDT protocol and FTP file transfer applications. Adapted from [21]. 11

2.2 Data and control paths in UDT protocol. Adapted from [23]. . . . 12

2.3 Secure chunk (a smaller part of the file) transfer process in the BioPeer. 14

2.4 Peer-to-Peer (P2P) file sharing process in BioPeer. 16

2.5 NAT Scenario. Adapted from [31]. 18

2.6 Hole punching algorithm with two clients. Adapted from [32]. . . 20

2.7 CDN implementation of BioPeer. 24

3.1 Performance test results in low latency network. 28

3.2 Performance test results in high latency network. 30

3.3 BioPeer mutli-client performance test result between server located in different countries. 31

3.4 Performance test result for 5GB file in detail. 31

List of Tables

3.1 Specifications of performance test servers. Adapted from [36]. . . 26

3.2 Locations and latencies of performance test servers. 27

Chapter 1

Introduction

High throughput sequencing (HTS) platforms have been developed in recent years that changed the way we do biological research. There are various HTS platforms with different strengths and biases. The most prominent HTS platforms are Illumina, Applied Biosystems SOLiD, 454 Life Sciences, PacBio, Oxford Nanopore, and BGI-Seq [1]. These high throughput sequencing technologies enable researchers to answer a wide range of biological questions by obtaining whole or targeted segments of genomes of individuals.

Developments in genome sequencing provide a means to generate very large amounts of data. Currently, there are more than 2,500 high-throughput sequencing machines worldwide that are manufactured by several companies, located in 1,000 sequencing centers in 55 countries [2]. These sequencing centers generate data from a few terabases to several petabases in a year [2]. Genome data size scales with the depth of coverage of the underlying sequence data, which is defined as the average number of reads that cover each position in the genome. For example, 40X coverage human genome (i.e. 40-fold redundancy to eliminate errors) size is approximately 100 GB in FASTQ format when compressed with gzip and 190 GB in BAM format. Most current platforms can sequence several genomes in a day. For example, a single Illumina NovaSeq instrument can read 48 human genomes in 2 days at 30X coverage. This translates to 720 TB data

in a FASTQ format or 1300 TB data in BAM format in a year. Considering the existing number of sequencers, it is easy to see that the yearly production of genomic data is beyond most other big data domains.

Most genome data are produced within several large-scale genome sequencing projects. For example, one of the major projects in this field is 1000 Genomes Project, which ran between 2008 and 2015. Through this project, one of the largest public catalogs of human variation and genotype data is created [3]. For its pilot project, 5 terabase pairs of sequenced data with size more than 12 TB in 38,000 files were produced. As of March 2012, 1000 Genomes Project created more than 260 TB of data in more than 250,000 files [4]. Other large-scale sequencing projects include 100,000 Genomes Project of the United Kingdom, Asian Genomes Project, AstraZeneca's 2 million genomes project, and Precision Genomes Project. 100,000 Genomes Project of the United Kingdom was launched in late 2012 and it aims to sequence 100,000 genomes. So far, 50,000 whole genomes have been sequenced in this project [5]. Asian Genomes Project's aim is to create reference genomes for Asian population and identify rare and frequent alleles associated with these populations. For this project, 100,000 Asian individuals will be used as samples [6]. Also, AstraZeneca, which is a pharmaceutical company, is developing another large-scale project in genomics field with Wellcome Trust Sanger Institute and Human Longevity. One of the aims of this project is to generate data from samples that belong to 500,000 participants. These participants have accrued over the past 15 years. The final aim of this project is to sequence 10 million human genomes. These genomes will be paired with medical records to understand the contribution of rare genetic variants to disease. So far, sequencing of 26,000 human genomes is completed in this project. It is expected to obtain around 5 petabytes of genome data at the end of the project [7]. Finally, Precision Genomes Project is developed by the United States. This project's aim is to analyze genomes of one million volunteers [8]. In summary, large-scale genome sequencing projects generate very large amounts of data.

Due to the size of the generated data, storage and transfer of the data became an important problem in genomics. Therefore, it is required to implement high-performance compression algorithms to reduce the size of genome data. Reducing the data size helps to reduce problems in both storage and distribution. First compression option is general-purpose compression tools such as gzip or bzip2. These compression methods can be applied to the files in both FASTQ and SAM format. However, they cannot efficiently reduce the files to manageable sizes in short amount of time [9]. In order to have a fast and better compression, several custom compression methods for genome data have been developed. Even with the best compression performance, 40X genome data still requires 20 GB for reads and 40 GB for alignments. It follows that data from a single NovaSeq instrument generated in a year will require 175 TB for reads. Therefore, specialized compression algorithms perform well on genome datasets generated by HTS platforms but data size still remains very large because of the original size of the data.

Besides the size of the genome data, another problem is distributing data among countries because most of the genome projects of contributors are located in different countries. For example, Great Ape Diversity Project, where over 20 TB of data was shared between several laboratories located in the United States, Germany, Denmark, Netherlands, United Kingdom and Spain [10]. Huge amounts of data transfer between collaborators located in different geographical locations are one of the most urgent issues to address in genomics.

The next section describes currently used methods to share genome data among project collaborators and their drawbacks.

1.1 Current Methods for Genome Data Sharing

Researchers in the genomics field use some methods to share genome datasets between project members. First, there are several repositories that are designed for biological data. Important examples are National Center for Biotechnology Information (NCBI) and European Nucleotide Archive (ENA). These repositories enable researchers to share their own data or download existing data. However, these repositories allow researchers to share their data in only two modes, public or private. The private mode could be used for a short time before making data publicly available. As researchers do not have a right to make their data accessible for only selected people, they have to use public mode for data sharing. However, researchers do not prefer to make their data available to everyone before publication. Therefore, public databases may not suit the needs of smaller research groups.

Besides the public databases, large-scale projects could use these cloud platforms like The Cancer Genomics Cloud (CGC) [11]. Like public databases, researchers could share their own data or work on existing data through cloud platforms. However, not all genome sequencing projects have access to large clouds. For example, CGC allows researchers to share or access data generated in only The Cancer Genome Atlas (TCGA) Project. Therefore, cloud platforms do not meet researchers' all needs about data sharing.

Another solution that is one of the widely used methods is transferring data with File Transfer Protocol (FTP) between collaborators. FTP is an application layer protocol implemented on Transmission Control Protocol (TCP) to transfer files between two hosts. FTP uses client/server architecture. As FTP uses TCP connection for data transfer, it provides reliable data transfer through acknowledgment messages. These messages show the success of data packet receiving. FTP also uses a congestion resolution algorithm, called CUBIC that is a modification of BIC-TCP. In CUBIC algorithm, congestion window growth function is defined in real time. Therefore, it is independent of Round-trip Time (RTT). However, when packet loss is detected, CUBIC algorithm decreases the window

size by 1/5 [12]. Therefore, reliability and the congestion control algorithm within TCP make FTP slower than other protocols in high latency networks [13]. Figure 1.1 shows that FTP receive rate decreases with the increase of Round-trip Time (RTT) because of congestion control algorithm of TCP. Tests with different file length (2, 4, 6, 8, 10 GB) shows that file transfer with FTP takes a long time in Figure 1.1 and Figure 1.2. In this test, FTP and UDT protocols are also tested with SSL (Secure Sockets Layer) protocol. Although SSL provides extra security to data transfer, it is causing speed decrease. According to the results, UDT with SSL is almost as fast as FTP without SSL [14]. Again, TCP does not perform well in slow and long distance networks because of its congestion control algorithm. Therefore, FTP is not an efficient solution for large file transfer.

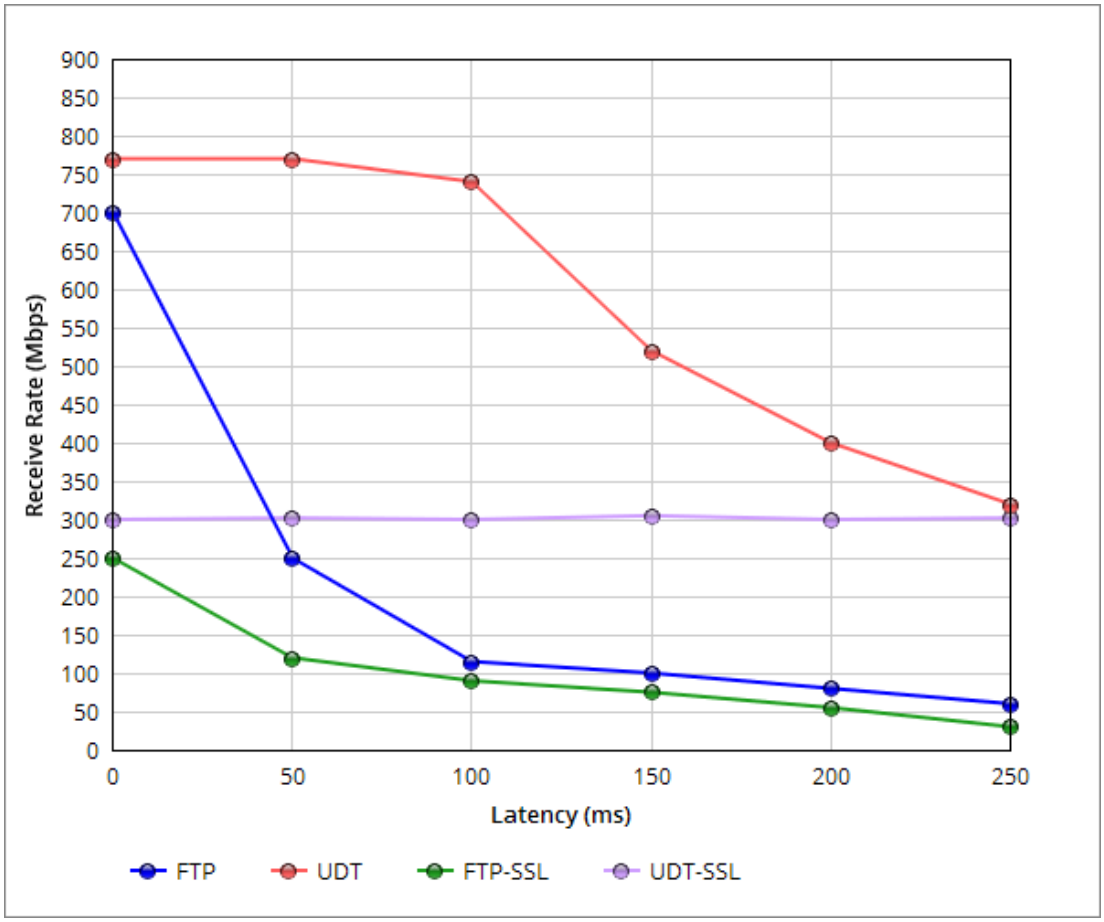

Figure 1.1: RTT comparison for FTP-UDT protocols in 1 Gbps link with 10GB file transfer. Adapted from [14].

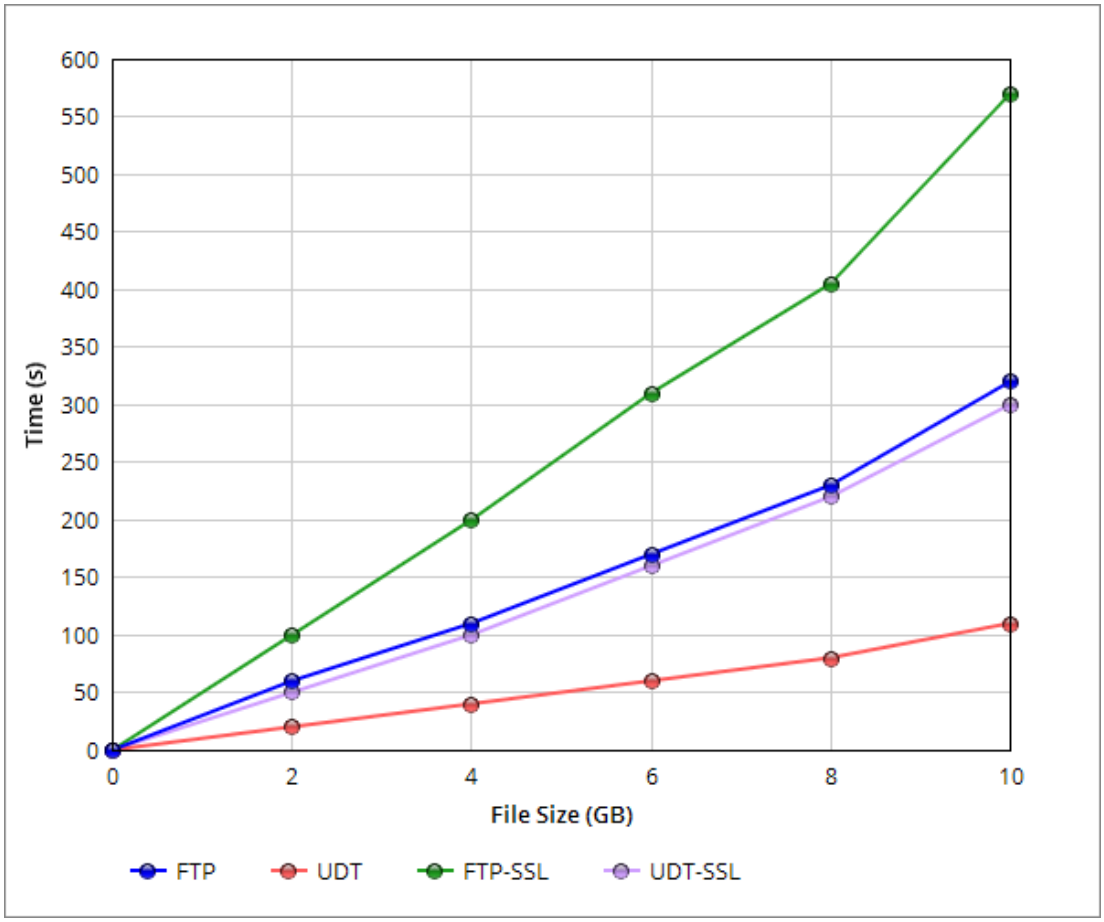

Figure 1.2: Performance Evaluation of FTP vs. UDT protocols with different file sizes. Adapted from [14].

Another solution for transferring large files is the Aspera FASP protocol. FASP uses User Datagram Protocol (UDP) in the transport layer [15]. Using their server/client tool, it is possible to transfer large files with up to 600 Mb/s speed. This is a tremendous improvement over the TCP-based FTP protocol. Large-scale projects such as 1000 Genomes Project and large sequencing centers such as Wellcome Trust Sanger Institute now use Aspera Software for file transfer. Although it provides the best data transfer speed, the main drawback of this solution is the cost. Most of the small research groups cannot afford to install and maintain a dedicated Aspera server and pay for the license costs, especially when their data transfer needs are not constant over time.

Another solution is transferring files through Tsunami Protocol. This protocol uses UDP for data transfer and TCP for control data that is used for managing retransmission requests and error rate information. Unlike other protocols, Tsunami does not send an acknowledgment of the received data packet. Instead of this acknowledgment, Tsunami sends a negative acknowledgment, which shows the packet loss periodically. Besides the reliability, it has flow control algorithm. Instead of sliding window algorithm, Tsunami uses inter-packet delay. If packet loss is smaller than the threshold, Tsunami increases the inter-packet delay exponentially. If packet loss above the threshold, Tsunami decrease inter-packet delay until a target rate has been met [16]. Through this protocol, data could be shared up to 600 Mbps [17]. Although Tsunami protocol achieves very high speed, it does not have a user-friendly graphical user interface (GUI). There is only command line interface (CLI) and C++ library of this protocol. And the current implementation of Tsunami allows connection between only two hosts. So, data could be transferred between only two peers. Therefore, it is difficult to use Tsunami as a data sharing platform between researchers.

Another solution is working on a central server instead of transferring files to all collaborators. Each researcher could upload their own data on a central server to store all required file together. Researchers could work on all files by connecting to the central server. However, as each collaborator works on a common server at simultaneously, very powerful machines are required to satisfy researchers' needs. Like Aspera Software, this solution requires a very high budget. Therefore, it is not a preferred solution for small research groups.

The last method which is commonly used to share data between collaborators is writing the data to external disks and circulating these disks using a courier service. This method requires more effort and time to store and share data.

As explained above, researchers use methods to share genome sequence projects between project members. However, currently used methods have some drawbacks. In this thesis, we propose an application, called BioPeer, as an alternative way of data sharing between collaborators. With this application, we aim to provide researchers to share genome data in a fast and secure way.

1.2 Contribution of the Thesis

In genomics filed, researchers need to share data between project collaborators located in diverse geographic locations. Due to file sizes of genome data and distances between project members, currently used file sharing methods do not satisfy the researcher's needs. As a solution for this issue, we developed an application, BioPeer.

Through this application, our first contribution is peer-to-peer (P2P) file-sharing architecture. Using the P2P architecture, we also implemented the parallel downloading feature to increase the throughput.

In P2P architecture, each peer requires being accessible through provided addresses. However, firewalls and Network Address Translation (NAT) based routers prevent to access peers. To address this issue, we implemented NAT-Traversal technique. Therefore even peers are behind the NAT-based routers or firewalls, they are informed about their public addresses. Through these public addresses, each peer would be accessible.

Our final contribution is the implementation of the content distribution network (CDN). To increase the performance of our application, in addition to project members, selected CDN nodes to store the files in encrypted mode. These nodes have rights to distribute the files to other peers. Through these CDN nodes, we increase the throughput of our application.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows. Chapter 2 describes the BioPeer application. In this chapter, we present features of BioPeer and motivation behind these features in detail. In Chapter 3, we evaluate BioPeer application according to test results. Finally, Chapter 4 concludes the thesis. In this chapter, we also propose future work.

Chapter 2

BioPeer: High Throughput UDP-based Peer-to-Peer Secure Data Transfer

This chapter explains BioPeer application's features in detail. Also, we discuss the motivation behind selected features of BioPeer.

2.1 Overview of BioPeer

Genome data size and distance between project collaborators create a problem about sharing data between project members in genomics field. This problem motivates us to develop an application, called BioPeer. BioPeer is a multi-platform desktop application that allows bioinformatics researchers to share their project data with collaborators in the project in a fast and secure way. BioPeer uses the UDP-based open source UDT protocol [18] for data transfer. Our application provides a peer-to-peer (P2P) file-sharing architecture similar to that of BitTorrent, where large files are transferred in chunks, and synchronized between

peers (i.e. collaborators) within the same project. Different from other P2P platforms, BioPeer also includes user authentication through the ORCID database (http://www.orcid.org) to ensure verified access. In addition, BioPeer encrypts files by using the Advanced Encryption Standard (AES) protocol. We use a unique 128-bit symmetric AES key for each chunk of the file to be transferred. To exchange of symmetric AES key between collaborators, BioPeer uses RSA (Rivest-Shamir-Adleman) cryptography. We generate unique public/private key pairs for each symmetric key transfer. Besides private projects, BioPeer also enables researchers to share their data publicly. Additionally, we solve network address translation (NAT) which is the common problem in P2P networks by applying hole punching algorithm. To increase the performance of the application, we also designed content distribution network (CDN). We implemented BioPeer with Java 8 to achieve platform independence, and it supports Linux, Windows, and OS X platforms.

2.2 Features of BioPeer

2.2.1 File Transfer Using UDP-based Data Transfer (UDT) Protocol

Most widely used protocols are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is a network communication protocol that provides an end-to-end reliable connection between hosts. Reliability means to ensure that data packet is received. TCP retransmits the data if any damage or loss occur during the transfer. To provide a reliability, TCP uses positive acknowledgment (ACK). Acknowledgment shows the success of receiving the data packet. If ACK is not received within the specified time, data is retransferred [19]. Through this property, TCP ensures that data packet is received by the corresponding host. Besides the reliability, TCP also offers congestion control mechanism because for a good performance network congestion is required to be prevented. TCP uses congestion window to control the number of data packets in

the network. As sending data process consumes a space in the congestion window, data packet transfer is allowed if there is free space in the window. Received ACK for data packet increases the amount of available space in congestion window. To manage window size, TCP uses an algorithm which is based on AIMD principle (Additive Increase Multiplicative Decrease). This principle causes a very slow increase of the congestion window [20]. Therefore, TCP does not perform well in slow and long distance networks. This drawback of the TCP causes long file transfer time for applications [13]. To solve this problem several variants of TCP, Scalable TCP, FAST TCP, HS-TCP, and BIC-TCP have been developed. However, Round Trip Time (RTT) unfairness or convergence makes these variants unsuitable for high-speed networks [21].

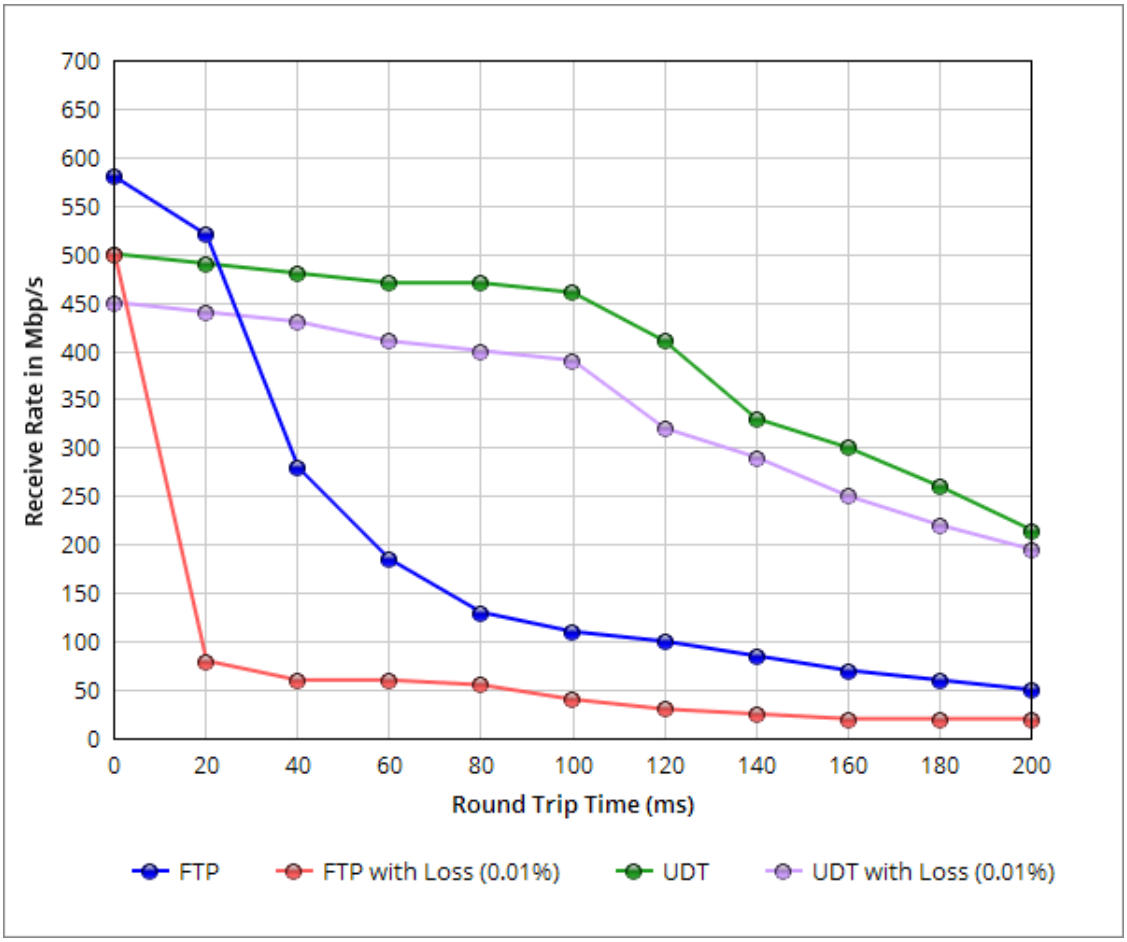

Figure 2.1: Comparison between UDT protocol and FTP file transfer applications. Adapted from [21].

Another popular data transfer protocol is User Datagram Protocol (UDP) [18]. UDP uses a simple connectionless communication mechanism that does not establish a connection before sending a packet. Since it is one-way communication, it cannot provide reliability. Therefore, it does not guarantee the delivery and protection of duplication [22]. However, reliability is important for file sharing process. Therefore, lost data packets are required to be retransmitted to obtain the entire file. Besides the reliability, UDP does not have congestion control mechanism. However, congestion control is needed to prevent congestion collapse and establish connection fairly [22]. Therefore, we did not use the basic form of UDP in our application.

To address both TCP and UDP limitations, UDP-based Data Transfer (UDT) Protocol was designed, which is built on UDP shown in Figure 2.2 [18]. UDT protocol transfers data packets through UDP port. As UDP is not reliable, UDT protocol implements reliability on UDP through two types of acknowledgments, selective positive acknowledgment, called ACK and negative acknowledgment, called NAK. ACK signals the successful receipt of packets. The receiver sends back ACK to the sender at constant time intervals as long as new packets continue to arrive. NAK is used when packet loss is detected. Sender retransfer the data packet if it receives the NAK packet. NAK is sent within a certain amount of time until the receiver gets re-transferred data [23]. Through these acknowledgments, although UDT protocol uses UDP connection, it provides reliable data transfer.

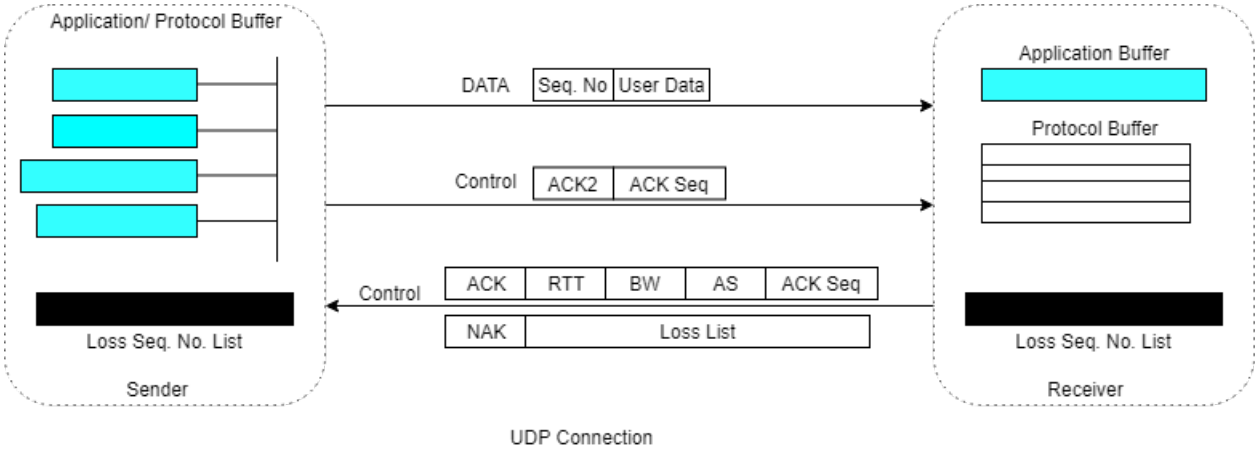

Figure 2.2: Data and control paths in UDT protocol. Adapted from [23].

Besides reliability, UDT protocol uses congestion control mechanism, based on the DAIMD (Decreasing AIMD Rate Control) algorithm [24]. Unlike TCP, data packet transfer is allowed if the number of unacknowledged packets does not exceed a congestion window. Although AIMD decreases the sending rate, DAIMD increases sending rate by decreasing additive parameter [24]. Through this congestion control, UDT protocol can utilize bandwidth efficiently and fairly within the long distance. As shown in Figure 2.1, UDT protocol performs better than FTP, based on TCP, through UDP's congestion control algorithm. Therefore UDT protocol achieves its goal through high throughput of bandwidth [21]. Drawbacks of TCP and UDP and UDT protocol's advantages make UDT protocol more suitable for our application.

2.2.2 Secure File Transfer

Genome data contains information about a person's hereditary characters. Therefore, they are sensitive and security is required when sharing. To protect data in our application from possible leaks, we encrypt the data using 128-bit Advanced Encryption Standard (AES) key before transfer. The AES algorithm is based on a symmetric key which is used for both encryption and decryption process [25]. However, using a single symmetric AES key for a file enables attackers to obtain the entire file through a key if any leak occurs in the network. To prevent this issue, BioPeer splits the files into chunks (smaller parts of the file) and transfers them separately. Thus, it generates a unique symmetric key for each chunk transfer to prevent obtaining file content through a single symmetric key.

Exchange of symmetric key between peers creates another issue in terms of security. Even when we use a unique symmetric key for each chunk transfer, these keys could still be collected maliciously due to the leak in the network during key exchange. To address this problem, we use RSA (Rivest-Shamir-Adleman) cryptography [26]. RSA algorithm also is the first implementation of Public-key cryptography. Through this method, we generate two different keys, public and private keys, which are linked with each other [26]. As only the public key is

shared with other hosts for encryption, only the private key owner could decrypt the message, which is encrypted with the public key. In the BioPeer application, a unique private/public key pair is generated and used for transferring each 128-bit symmetric AES key.

Besides these cryptographic precautions, we do not allow everyone to use our application. To login BioPeer, we use ORCID database platform for user authentication [27] because our target group consists of researchers.

Figure 2.3 shows secure chunk transfer process used in BioPeer. First, the receiver generates 128-bit AES symmetric key for current chunk transfer. To exchange this symmetric key, the sender creates RSA public/private key pair. Receiver encrypts the AES symmetric key with the sender's public key and sends it to the sender. As only the sender has a private key that depends on the public key, the only sender can decrypt AES symmetric key. Then the current chunk of the file is encrypted with the symmetric key and transferred to the receiver. Since the receiver has the corresponding symmetric key, data could be decrypted.

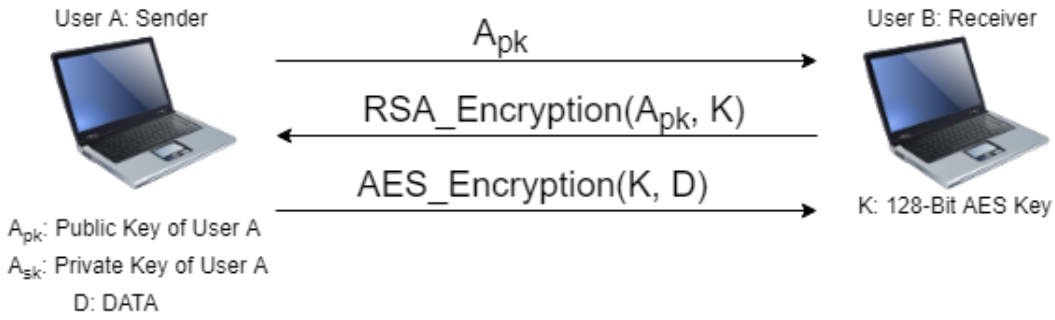

Figure 2.3: Secure chunk (a smaller part of the file) transfer process in the BioPeer.

2.2.3 Peer-to-Peer (P2P) File Sharing

A client/server network is a distributed network which consists of one server with higher specifications and multiple clients with lower specifications. A server is a central unit which is responsible for providing content and service. The client could request content or service from the central server without sharing resources

with other clients [28]. Genome sequence projects contain very large amounts of data and a large number of collaborators. As there is a single server for responding multiple clients, several requests from clients to a server may cause increased load on the server. Also, if the central server fails, the entire network is affected because the only central server is responsible providing content and service. Since there are very large amounts of data and several users, very powerful machines are required. Therefore high cost is another problem in client/server architecture. Besides the cost of the server, researchers do not prefer to store genome data on a central server because of security and privacy of the genome data. Due to these drawbacks, client/server architecture is not suitable for the problem that aims to solve.

As an alternative to client/server architecture, peer-to-peer (P2P) networks were proposed as collections of heterogeneously distributed users which are connected by the Internet [28]. In this architecture, there are no central servers to provide content. Each user acts as both a server and a client. Thus, each peer is responsible to distribute content. Therefore, each user should be able to access all other users for sharing the data. As there are several users who act as a server, this architecture enables clients to download parts of the data in parallel. Through parallel download, the P2P architecture provides higher throughput. Also, existence of several clients that act as senders prevents bottleneck in the system. There are usually multiple clients available for responding to the requests. Unlike client/server architecture, failure of any sender does not affect the entire system due to the existence of several servers. Peers in the system could find another server to request content instead of the failed server.

Figure 2.4 shows P2P file sharing process in BioPeer. Files are transferred in chunks and synchronized between peers. Each user has a right to share downloaded chunks with other collaborators defined in project. Through P2P architecture, peers could download chunks of a file in parallel. As shown in Figure 2.4, User B downloads different chunks of a file from User A and User B simultaneously.

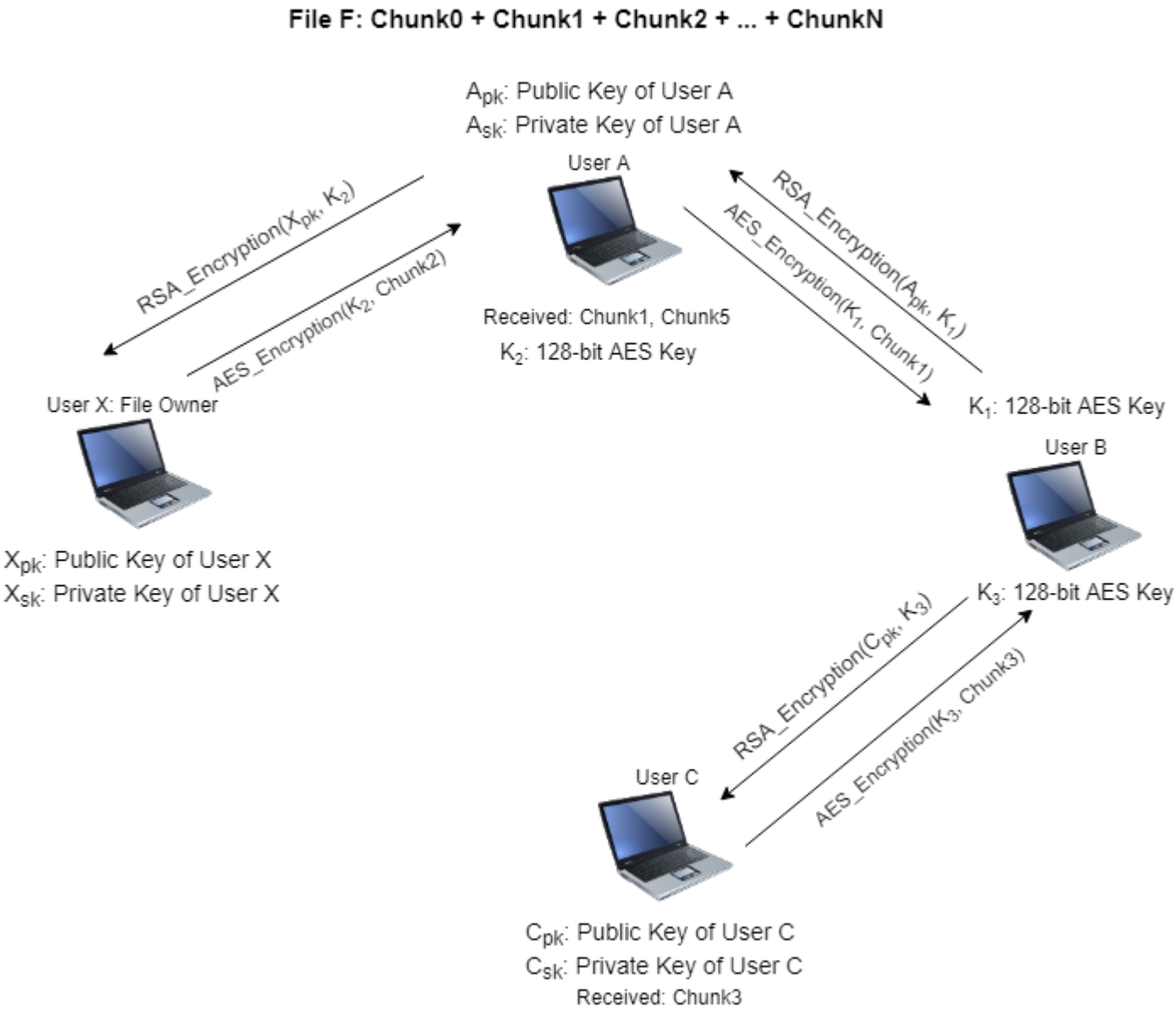

Figure 2.4: Peer-to-Peer (P2P) file sharing process in BioPeer.

2.2.4 NAT-Traversal

Firewalls are commonly used to improve the security of shared networks like companies, institutions, or universities. It protects networks from external attacks through preventing unauthorized access [29]. Some firewalls only allow connections to be initiated from the private network unless it is configured specifically. Firewalls commonly deny access to ports associated with data streaming applications like BitTorrent. Therefore hosts in firewall protected networks might be unreachable.

Network address translation (NAT) is a system which allows multiple hosts to use a single public IP address. Private IP address is mapped to a public IP address by NAT-based routers. These routers exchange also public and private IP addresses in outgoing and incoming messages. Through mapping process on IP addresses, private IP address of the host behind NAT gateway becomes unreachable from public Internet [30].

Firewalls and NAT cause problems in P2P networks. As peers have roles as both a server and a client in P2P networks, each peer needs to be reached directly from IP address in the network for communication. As firewalls and NAT-based routers exchange the hosts' addresses, peers might be unreachable. Therefore, we needed to solve this problem to make our application useful for all users regardless of their network setup.

The simplest solution for NAT and firewall is configuring the router manually. However, this solution is inconvenient and requires administration privileges. However, not all hosts in the network have right to access router. Therefore we need to implement appropriate NAT traversal method to BioPeer.

Possible NAT traversal methods are:

- Universal Plug and Play (UPnP)
- Hole Punching
  - Simple Traversal UDP Through Network Address Translators (STUN)
  - Traversal Using Relay NAT (TURN)
  - Custom UDP hole punching implementation

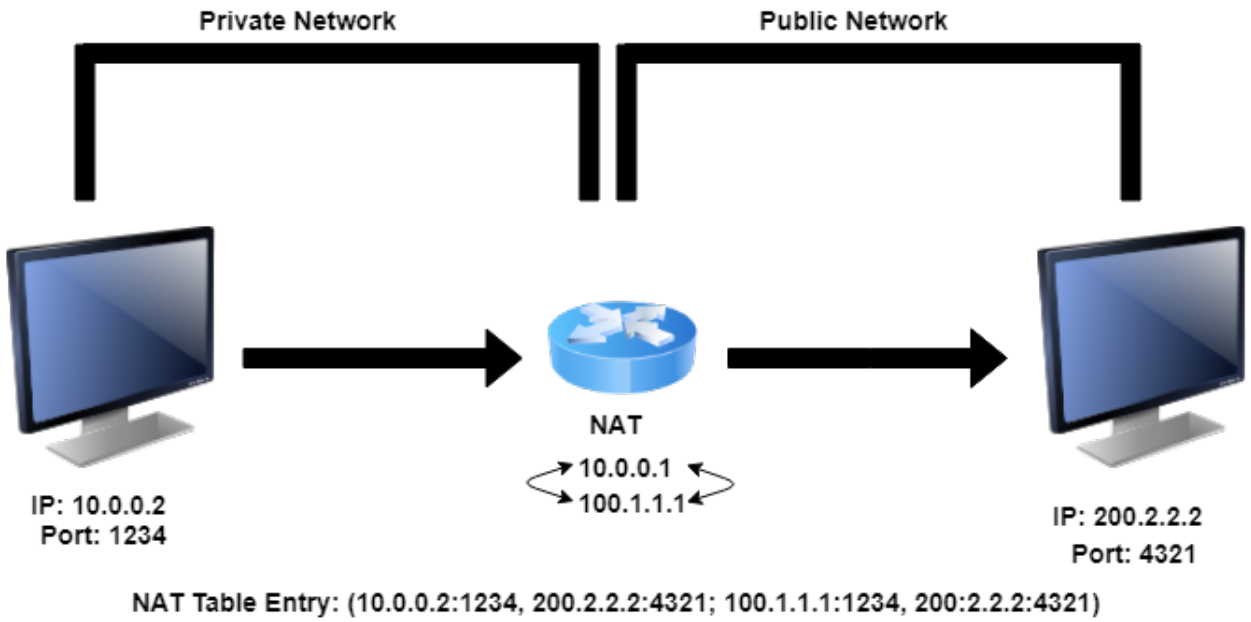

Figure 2.5: NAT Scenario. Adapted from [31].

In this section, we discuss these four NAT traversal methods whether they are suitable for our application or not.

2.2.4.1 Universal Plug and Play (UPnP)

UPnP is architecture to enable hosts that are behind the UPnP-enabled NAT gateways to communicate directly. In the connection process, the application could detect the existence of an UPnP-enabled NAT device. UPnP device can

learn public IP address of the host that is behind the UPnP-enabled NAT gateway. Program configures UDP and TCP port mappings for data exchange between external ports of the NAT and internal ports [32]. Packets are forwarded from external ports in the public address to internal ports. Therefore it is an automatic port forwarding process. To efficiently use this protocol, the router should support UPnP.

BitTorrent, which is the most commonly used P2P data sharing application, uses UPnP protocol as a NAT traversal technique. However, UPnP protocol requires router support. Like manual port forwarding, administration privileges are also required to enable UPnP protocol in the router. Because of these requirements, we did not use UPnP in BioPeer

2.2.4.2 Hole Punching

Hole punching is a technique to establish end-to-end communication in P2P networks shown in Figure 2.6. Through this technique, the application does not need to know the existence of NAT. In this technique, there is a relaying server that can be reached by all clients. Clients send registration message that contains public IP address and port to the server. Through this registration message, the server informs peers about their public IP address and port. Through public addresses, peers could relay data to each other directly. Therefore in this method, there is no need to modify the NAT or firewall [32].

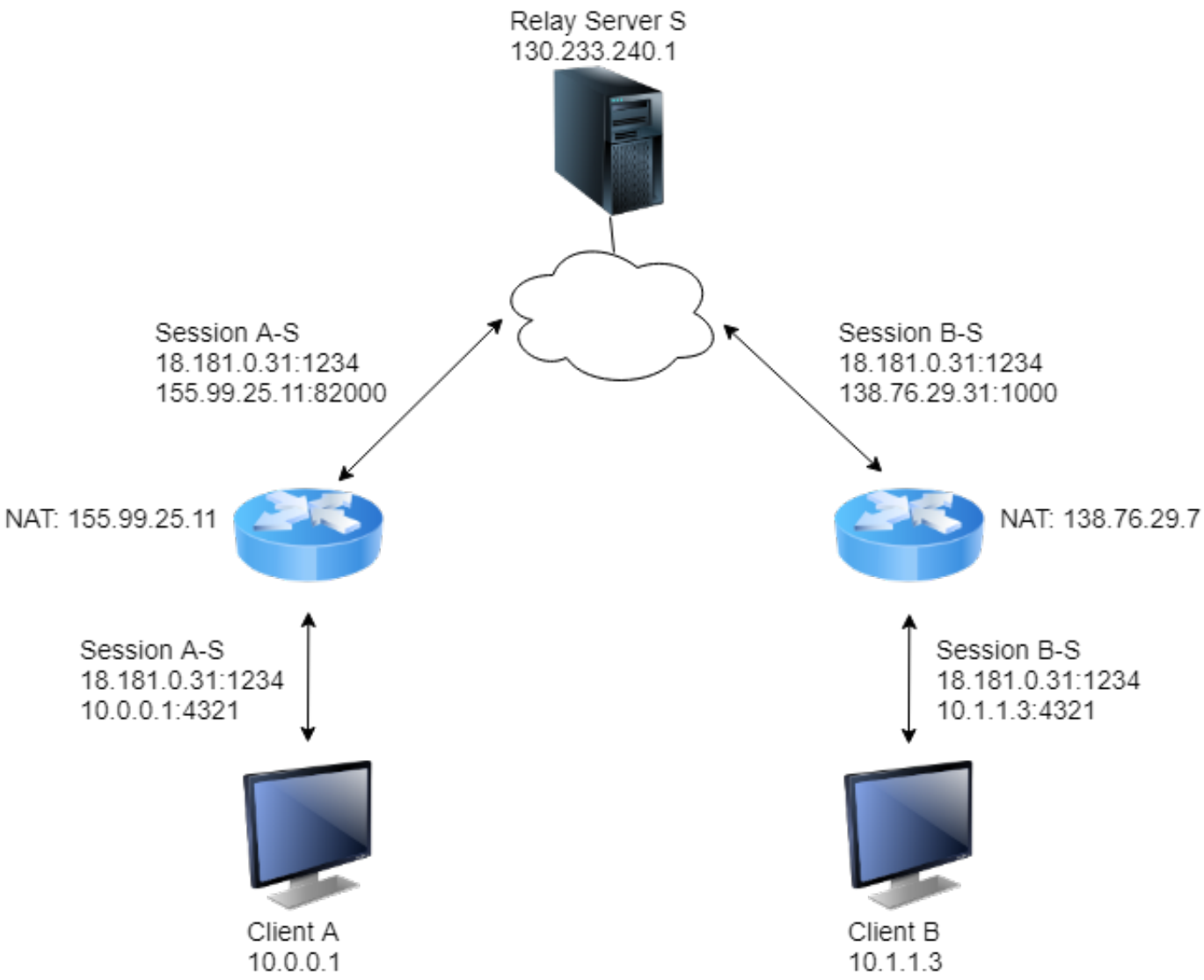

Figure 2.6: Hole punching algorithm with two clients. Adapted from [32].

There are several different implementations of hole punching algorithm:

2.2.4.2.1 Simple Traversal UDP Through Network Address Translators (STUN)

Simple Traversal UDP Through Network Address Translators (STUN) is one of the hole punching implementation. STUN is a client/server based protocol. STUN clients can learn IP address and port used on NAT from a third-party

network server, STUN server, via a binding request. The STUN server could identify the last NAT modified source address and port. Then the server sends IP address and port behind NAT to the STUN client. However, this method does not work with symmetric NAT. The fact that symmetric NAT is being used commonly makes STUN technique unpopular [33]. The unpopularity of STUN method causes lack of active development of Java libraries for STUN. Therefore, we could not implement this protocol to BioPeer.

2.2.4.2.2 Traversal Using Relay NAT (TURN)

TURN protocol is an extension of STUN. If STUN method fails, TURN protocol is used as a backup method for NAT traversal. There is an external server which is called TURN server. The client obtains a relayed transport address which contains IP address and port from TURN server. Peers relay data packets to the client by using transport address of client and server [34]. So besides the obtaining address information, TURN server is used for data transfer. Using server for data transfer makes TURN technique unsuitable for BioPeer because genome data are not preferred to relay to or store on external servers in terms of privacy and resource issue.

2.2.4.2.3 Custom UDP hole punching implementation

In this implementation, similar to STUN and TURN, there is an external server to obtain address information. This server listens specific port continuously. Clients that are involved in data transfer process open UDP port to connect external server. They send registration that contains public addresses. The external server sends public address information to clients. Also, clients are informed about the public addresses of another peer in the transfer process. Therefore, this implementation enables two clients to communicate directly with each other using the peer's public IP and port.

We preferred this method as NAT traversal technique in BioPeer since it does not

require any manual configuration by the user. It automatically finds an available port and uses it in the data transfer.

2.3 Content Distribution Network (CDN)

Performance of web connection is measured with Internet service quality which represents content delivery time [35]. Sharing content in a P2P network increases the service quality through parallel upload and download. However, even with a P2P network architecture, distributing the very large amounts of data causes a problem. Few numbers of clients that have content could not serve large number of requests in a short time. This situation decreases the performance of the application. Also, it prevents peers to utilize the full bandwidth of their network. For example, in genomics, some of the projects consist of huge amounts of data. 1000 Genomes Project contained more than 260 TB of data [4] and researchers expect to generate approximately 5 petabytes of genome data for AstraZeneca's 2 Million Genomes Project [7]. These large-scale genome projects include a very large number of collaborators. Distributing these huge amounts of data between project contributors within the P2P network could cause a problem due to limited network bandwidth.

Implementation of the content distribution network (CDN) is an effective solution for the performance problem. CDN is a distributed server network that stores copied content and increases the delivery. In this network, there are several nodes that store the copies of the content. When original servers in the system do not satisfy the requests, these external nodes distribute content to peers. Therefore, through CDN, the network provides lower latency and higher transfer rate, which increase the performance of P2P architecture [35].

2.3.1 CDN Implementation of BioPeer

To increase performance, we implemented CDN algorithm into BioPeer. Except for the project collaborators, selected CDN nodes that we treat as untrusted users store chunks of files to relay to clients in the project. When there are no collaborators in the project that are available to provide content peers request data from these external CDN nodes. However, security of the genome data is an issue when distributing to CDN nodes as genome data is sensitive. CDN nodes cannot access the content of the data as they are not project collaborators. To address this problem, project owner generates a unique 128-bit AES symmetric key for each chunk of the file. When any user exchanges the data with these CDN nodes, they need to use these unique symmetric keys for encryption or decryption process. Therefore, each peer within the same project needs to have unique symmetric keys that are generated for transfer with CDN nodes. We use RSA cryptography to exchange these symmetric keys between project collaborators to store their embedded SQLite databases. Project users download these symmetric keys before downloading files. Also, as CDN nodes are not assumed to be project collaborators, we do not share the symmetric keys with these untrusted nodes to prevent obtaining data content.

Project members encrypt chunks of the file with AES encryption algorithm with corresponding keys in SQLite database before transferring to CDN nodes. As symmetric keys are not shared with these untrusted nodes, they cannot decrypt and see the original content. In addition to these cryptographic precautions, CDN users are not allowed to store all encrypted chunks of a file because if these untrusted nodes obtain all AES symmetric keys maliciously, they could obtain the entire file. To prevent this situation, CDN nodes have rights to keep only a few numbers of chunks.

In figure 2.7, we showed that when project collaborators request to download a file chunk, a relative CDN node is selected in order if there are no available project users. Users who are invited to the project get the list of symmetric keys from the project owner and store these keys in a local SQLite database. When

a collaborator downloads a file chunk from CDN nodes, this chunk is decrypted with corresponding AES keys from embedded SQLite database.

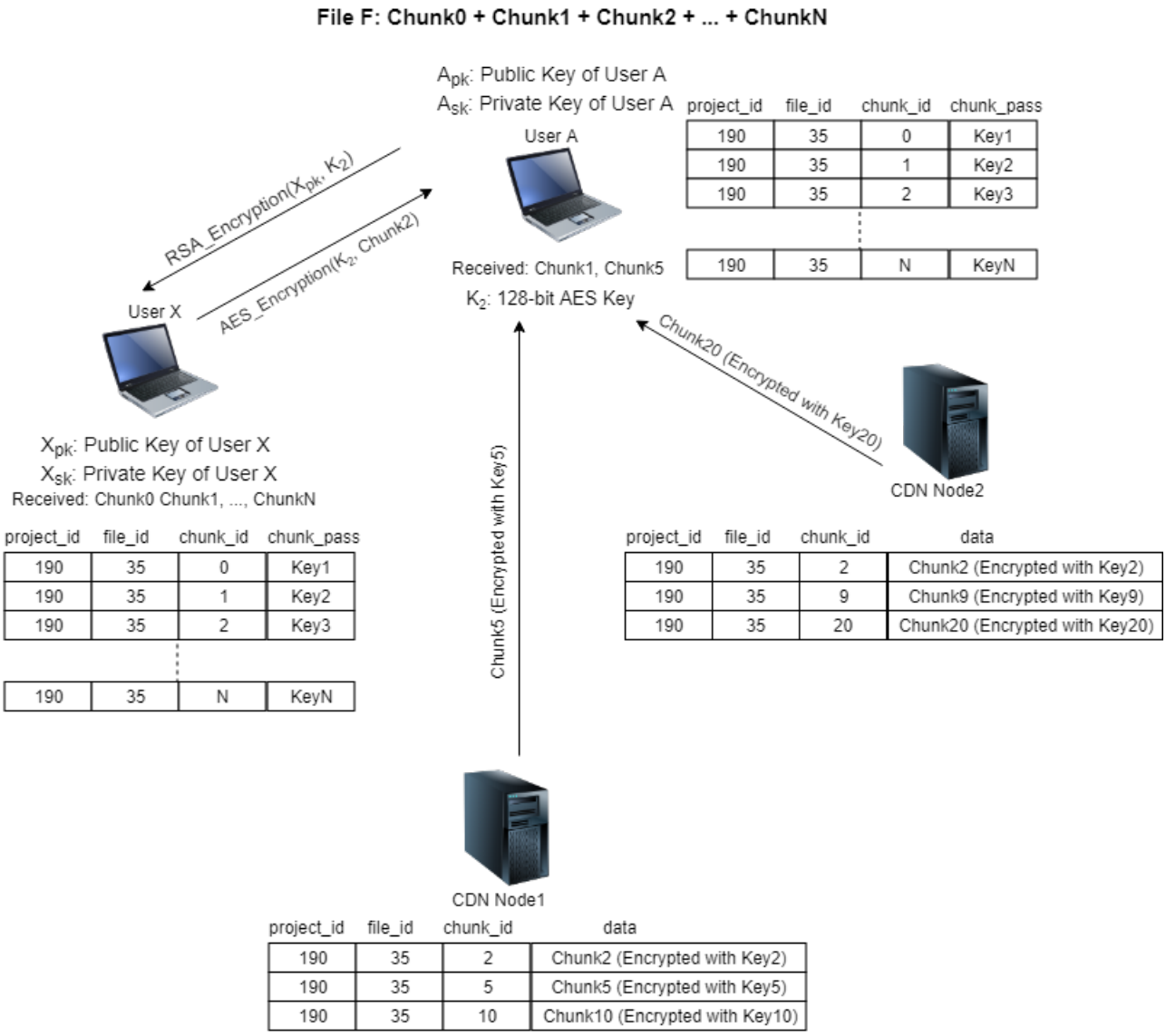

Figure 2.7: CDN implementation of BioPeer.

2.4 Sharing Data in Public Projects

In genomics field, some of the important projects like the 1000 Genomes Project host huge data and share them publicly. The researcher could download files of the 1000 Genomes Project from the links on its website [3]. The 1000 Genomes Project uses Aspera software to distribute the data to the researchers. BioPeer also enables researchers to download files from the public projects.

BioPeer allows project owners to make their project public during the project creation process. In public projects, only the project owner can modify the project by adding or removing files. Other users could only download and share data with other collaborators. Researchers can find the public projects by searching the project identification number.

As most of the public projects host huge amounts of data, trusted users who download project files may not be able to distribute files efficiently. Apart from users who downloaded files from public projects, public CDN nodes help us to distribute data to other researchers.

Chapter 3

Performance Evaluation of BioPeer

We conducted our performance experiments using virtual servers available at Digital Ocean Cloud Services [36]. We give the server specifications in Table 1.

Table 3.1: Specifications of performance test servers. Adapted from [36].

Virtual Machine Type	Standard Droplet
CPU	4 vCPUs
RAM	8 GB
Storage	160 GB SSD
Network	1 Gbit/s-Port
OS	Ubuntu 16.04.4 x64
Location	London, Bangalore datacenter

For our performance tests, we used test files with different lengths [37]. These files contain randomly generated bytes. We transferred the same files via different types protocols within the same environment. We used different protocols to compare BioPeer performance with these alternative protocols that are based on either TCP or UDP. We compared BioPeer in terms of performance with different UDT protocol implementation, Secure Copy Protocol (SCP) and Tsunami protocol [38]. BioPeer, UDT protocol, and Tsunami protocol are based on UDP, and SCP is based on TCP.

We considered two options to implement UDT protocol to BioPeer. These are 1) a Java implementation of UDT protocol implemented by Johannes Buchner [39] and 2) a Java wrapper for native C++ implementation of UDT protocol written by Barchart, Inc [40]. We used these two different implementations of UDT protocol for performance evaluation tests to select best-performing library. As we use UDT protocol written by Barchart, Inc in BioPeer, it has same performance results with UDT protocol implemented by Barchart, Inc.

Another protocol that is used for performance evaluation test is Tsunami protocol. In order to use Tsunami protocol for performance evaluation, we used portable binaries suitable for our testing environment [41]. First, we started a Tsunami server for serving test files to download. In the client machine, we started a Tsunami client, connected to the Tsunami server and downloaded test files in order.

SCP is the last protocol in our performance evaluation. Ubuntu 16.04.4 distribution comes with SCP program installed and ready to use. We transferred the test files using SCP command in order.

Table 3.2: Locations and latencies of performance test servers.

Server 1	**Server 2**	**Latency**
London	London	0.340 ms
London	Bangalore	140 ms
London	New York	69 ms
London	Singapore	168 ms
London	Amsterdam	8 ms
London	Toronto	83 ms

The first performance test was performed on a low latency network. We used two virtual servers based in London, United Kingdom with 0.340 ms latency (Table 3.2). We transferred files with different sizes (100 MB, 512 MB, 1 GB and 10 GB) between these virtual servers. Figure 3.1 shows the performance test results of BioPeer and other protocols in a short distance network.

First performance test results in Figure 3.1 show that TCP based protocol, SCP, performs better than UDP based protocols in short distance networks. In TCP,

during data transfer, ACK messages that show the success of packet transfer is used. Transferring these ACK messages makes TCP slower than UDP. However, in low latency networks, time for transferring ACK messages is negligible. As originally, TCP was designed for short distance networks with low latency [42], it is expected to see that SCP performs better than UDP based protocols. Therefore, in low latency networks with no packet loss, it is not preferred to use UDP based protocols for reliable data transfer.

Figure 3.1: Performance test results in low latency network.

In the next step, we made a more realistic performance test where clients are distributed around the world because most of the genome project contributors are located in geographically diverse locations. Therefore, BioPeer aims to transfer data between researchers in the long-distance network.

We transferred files with different lengths (100 MB, 512 MB, 1 GB and 10 GB) in a long distance network, which is BioPeer's target group. For this experiment, we used virtual servers based in London, United Kingdom and Bangalore, India with 140 ms latency.

Although researchers currently use FTP that is based on TCP to transfer genome data, experiment results show that TCP based protocols perform poorly in high latency network. Time for transferring ACK messages makes TCP based protocols slower than UDP based protocols in this network. Therefore, TCP performance is inversely proportional to Round Trip Time (RTT) of the network. So, TCP is not preferred for transferring huge data in long distance networks.

Although Johannes Buchner's UDT protocol implementation performs well in low latency networks, its performance diminishes in high latency networks. We believe this protocol is not optimized for real-world situations. In addition, the code has not been maintained for a long time. Therefore, we decided to use Barchart's UDT library in BioPeer that is faster than Buchner's UDT implementation.

Although latency increases, performances of Tsunami and BioPeer remain same (Figure 3.2). Therefore, results show that even if there is a high latency, Tsunami and BioPeer could utilize the network bandwidth efficiently. However, although both UDT and Tsunami are based on UDP, there is a significant difference between performances. There are several reasons for having performance difference between Tsunami and UDT protocols. The most important reason is lack of tolerance to packet losses. Packet loss rate has an impact on UDT throughput but it does not affect Tsunami greatly. However, when packet loss is more than 1% throughput of Tsunami drops quickly [17]. Therefore, the performance of the Tsunami is better than UDT protocol when packet loss is small.

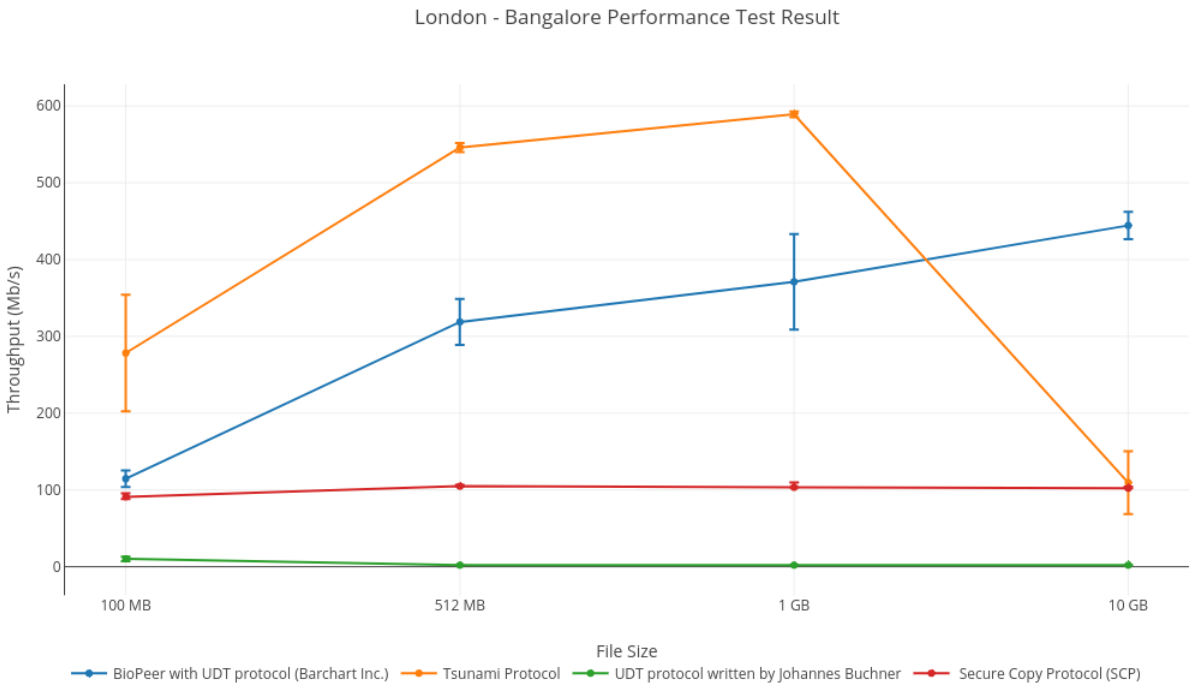

Figure 3.2: Performance test results in high latency network.

For previous performance evaluation tests, we used single BioPeer client to download the file. However, BioPeer enables clients to download in parallel through peer-to-peer architecture. To test this functionality, we used 6 virtual servers based in London in United Kingdom, New York in United States, Singapore, Amsterdam in Netherlands, Toronto in Canada and Bangalore in India, which acted as BioPeer clients. Latencies of these servers are shown in Table 3.2. Bandwidth limit of these servers are approximately 1Gb/s. The virtual server in London acted as a BioPeer client that downloads the file. Other virtual servers acted as content provider. We transferred 1 GB, 5 GB, 10 GB, and 50 GB files between these virtual servers. As shown in Figure 3.3 BioPeer client could download file from several peers in parallel with 1000 Mb/s. Through increasing the number of clients, BioPeer could use full bandwidth in the network.

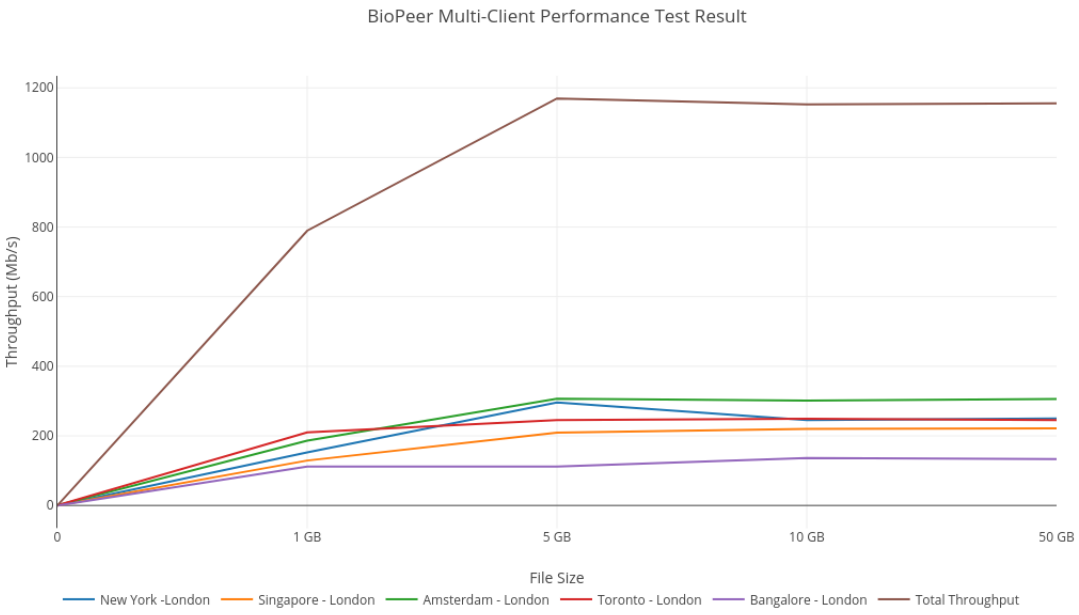

Figure 3.3: BioPeer mutli-client performance test result between server located in different countries.

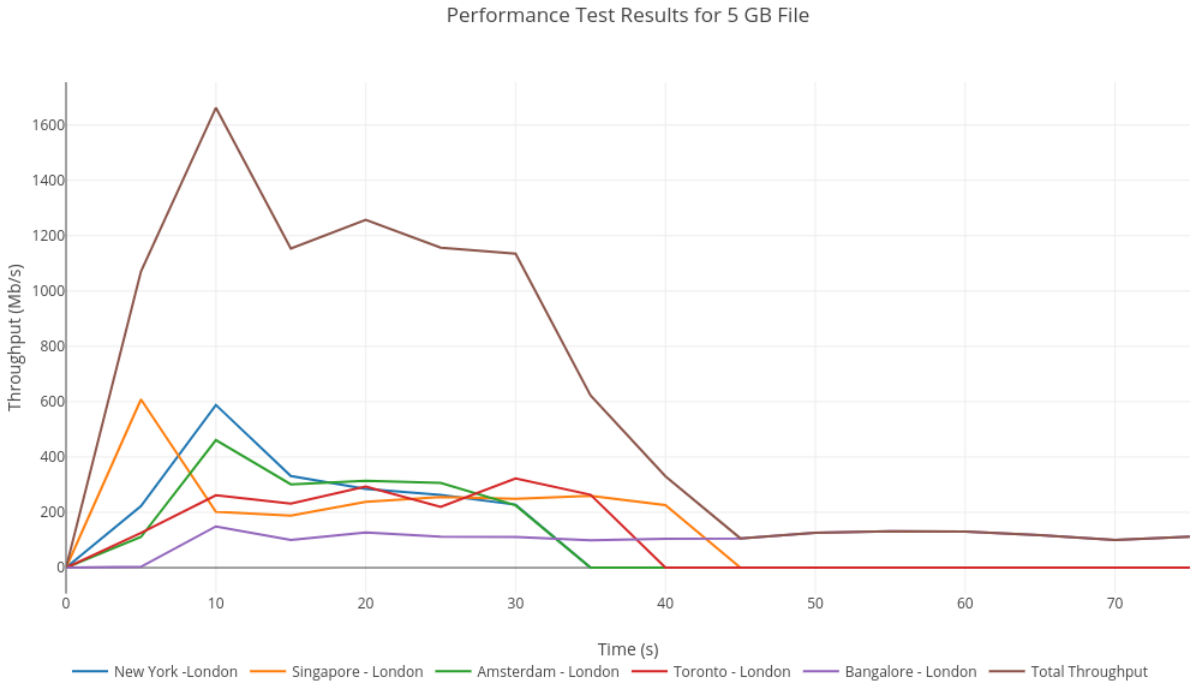

Figure 3.4: Performance test result for 5GB file in detail.

Figure 3.4 shows transferring process of 5 GB file in detail. Each peer in located different location sent a chunk of the file that is 1 GB. As seen in the speed chart, different peers send the data with different speeds. The receiver server is able to utilise the available bandwidth and download with the maximum speed while downloading chunks from multiple clients. Total downloading speed decreases by 80% when most peers complete the download and the slowest server is left.

Chapter 4

Conclusion & Future Work

4.1 Conclusion

In this thesis, we introduced the problem on transferring huge amounts of data between collaborators located in different geographical locations. Currently used transfer methods have some drawbacks like cost, lack of user interface, speed, and privacy. To address these issues, we developed an application called BioPeer.

BioPeer has a peer-to-peer software architecture for file transfer. BioPeer transfers files in chunks, smaller parts of a file, via the UDT protocol. As genome data are sensitive, we require secure file transfer to protect shared data. To provide secure file transfer, BioPeer encrypts files with Advanced Encryption Standard (AES) keys before transmitting. Also, for increasing our performance, we implemented content distribution network (CDN) infrastructure to our application. In this feature, in addition to the project collaborators, external clients, CDN nodes, store the encrypted chunks of the files without having access to the encryption key. Finally, during the development process, we faced a problem. Peers could be unreachable because of firewalls and NAT-based routers. To solve this problem, we implemented NAT-traversal technique to our application. In summary, BioPeer provides a secure and fast file transfer method to the researchers.

After development process, we performed performance evaluation tests of the BioPeer using three directions; high latency network, low latency network and transfer file between multi-clients of BioPeer located in different locations. We also compared BioPeer with alternative protocols on high and low latency networks. These protocols are UDT Java library written by Barchart Inc., TCP based SCP protocol, UDP based Tsunami protocol and UDT Java library written by Johannes Buchner. For each test, we used files with different sizes that contain random data [37].

In the short distance network with low latency, SCP performs better than other protocols owing to TCP's advantages in low latency networks. However, we aim at BioPeer to be used in long-distance networks because most of the genome project contributors are located in different locations. Therefore, we made the more realistic performance test on high latency networks. Test results in long distance network with high latency showed that UDP based protocols; UDT and Tsunami perform better than TCP based protocols.

For previous tests, we used single BioPeer client. However, our application enables users to download data from multiple peers in parallel. Therefore, we did the same performance test with multiple BioPeer clients. This performance test showed that downloading the file from multiple clients with BioPeer allows us to obtain high throughput transfer and use most of the available bandwidth. Through increasing the number of clients, we can increase the performance of our application.

4.2 Future Work

For data transfer, TCP and UDP are most widely used protocols. However, UDP is an unreliable protocol. And, our performance test results show that TCP performs poorly in high latency networks that is our target group. As speed and reliability are important in our application, we used UDP based data transfer (UDT) protocol in BioPeer to transfer files between project collaborators

instead of TCP and UDP. While making performance tests, to compare with our application's performance, we used another UDP based protocol, Tsunami. These performance test results show that Tsunami protocol performs better than UDT protocol in both low and high latency networks. Tsunami protocol uses UDP for data transfer and TCP for control data transfer [16]. Performance test results show that files could be transferred via Tsunami protocol up to 500 Mbps even on a high latency network. Tsunami protocol also provides the reliability. Through Tsunami and parallel download, BioPeer could be faster than the current implementation. However, Tsunami protocol is implemented in C. To be able to use Tsunami protocol in BioPeer, we need a Java implementation.

There are several options for using Tsunami within BioPeer application. The first option is to re-implement the protocol from scratch using Java. Java implementation requires a significant engineering effort. It is not a preferable solution due to maintenance issues. There are several methods for making native libraries accessible from Java applications without implementing the whole library. Java has an official interface named Java Native Interface (JNI) to interoperate with libraries written in other programming languages like C and C++. Using JNI, the developer adds a layer between Java and the native application by implementing an interface for function and object definitions. So, Java application can call functions of the native application. However, JNI implementation still requires maintenance for keeping interfaces synchronized with the native implementation. There is also a community-developed library named Java Native Access (JNA) to provide Java applications to access native functions using JNI [43]. This library allows developers to implement interfaces with a minimum effort without implementing the boilerplate code. Therefore, maintenance of keeping interfaces up to date with the native implementation does not become an issue. By implementing one of these options, we could use Tsunami protocol instead of UDT protocol.

Another future work is improving our Content Distribution Network (CDN) algorithm. As limited peers in large-scale projects could cause congestion on project collaborators. Therefore, we implemented the basic principles of CDN algorithm to BioPeer. In this algorithm, in addition to project members, CDN nodes also store the chunks of the files. These nodes have a right to distribute chunks of the

files to project collaborators. In our implementation, we have a list that contains nodes that have required chunk of the file. We select CDN nodes sequentially from this list. However, this kind of node selection algorithm does not utilize the total capacity of the network because same nodes could be selected while they are busy. There are several proposed solutions to the node selection problem in order to use most of the available network bandwidth [44]. One of the difficulties in CDN is selecting a good node to request file transfer. Good node is the node that sends requested chunk of the file to client faster than the ordinary client. Not selecting good node could cause to decrease the performance due to latency [44]. Therefore, we need to improve the selection of CDN node algorithm. In order to test CDN node selection algorithms, the platform needs to have more than a couple of users. Since this improvement is in our future work list, we haven't tested and evaluated the current node selection implementation.

In BioPeer, we divide the files into smaller chunks with a fixed size. At the end of each file download operation, the client validates the downloaded file by comparing the MD5 hash with the original file hash. If the client receives a damaged or altered block, it has to download the entire file again because we don't know which chunk is faulty. This method can be more efficient by using a more suitable hashing algorithm and file segmentation. Instead of having fixed-size chunks, we can have smaller size blocks and download these blocks as long as the connected peer has available blocks to download. Instead of calculating the MD5 hash of the whole file, Merkle tree can be used for data validation. Merkle tree algorithm allows us to verify the contents of large data structures efficiently [45]. So, in case of a damaged or altered block, clients don't have to download the entire file again. The damaged or altered block can be downloaded again and validated immediately using other verified blocks.

Another improvement can be made to download randomization. As in BitTorrent protocol, clients can download random parts of the file instead of downloading the file blocks sequentially. For example, two clients are downloading a file from a project owner, these clients will be able to share part of the file with each other if they download random parts of the file. In the current implementation, they will download the same chunks from the project owner. So, they are not able to

share downloaded chunks with each other since they have the same part of the file.

Finally, our application's target user group is researchers in genomic filed. In this filed, genome sequencing projects consist of files with very huge size. Transferring huge amounts of data could take a long time even with best performing data transfer protocol. Reducing the file size decreases the data transfer time. Therefore, before transferring the files, it is important to compress files with efficient compression algorithms to reduce the file size. In genomics, the file could be compressed by 80% with specialized compression methods that are developed for genome data. Reduction in this amount decreases the data transfer duration efficiently. Therefore, before transferring process, we could identify the uncompressed files and compress them with a special genome data compression algorithm to make transfer faster.

Bibliography

[1] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, *et al.*, "The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data," *Genome research*, vol. 20, no. 9, pp. 1297–1303, 2010.

[2] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: astronomical or genomical?," *PLoS biology*, vol. 13, no. 7, p. e1002195, 2015.

[3] "1000 genomes - a deep catalog of human genetic variation." http://www.internationalgenome.org/. [Online; accessed 2-February-2018].

[4] L. Clarke, X. Zheng-Bradley, R. Smith, E. Kulesha, C. Xiao, I. Toneva, B. Vaughan, D. Preuss, R. Leinonen, M. Shumway, *et al.*, "The 1000 genomes project: data management and community access," *Nature methods*, vol. 9, no. 5, p. 459, 2012.

[5] "Genomics england - 100,000 genomes project." https://www.genomicsengland.co.uk. [Online; accessed 2-February-2018].

[6] "Genomeasia 100k." http://www.genomeasia100k.com/. [Online; accessed 2-February-2018].

[7] H. Ledford, "Astrazeneca launches project to sequence 2 million genomes," 2016.

[8] F. S. Collins and H. Varmus, "A new initiative on precision medicine," *New England Journal of Medicine*, vol. 372, no. 9, pp. 793–795, 2015. PMID: 25635347.

[9] F. Hach, I. Numanagić, C. Alkan, and S. C. Sahinalp, "Scalce: boosting sequence compression algorithms using locally consistent encoding," *Bioinformatics*, vol. 28, no. 23, pp. 3051–3057, 2012.

[10] "Gagp - home." http://biologiaevolutiva.org/greatape/. [Online; accessed 2-February-2018].

[11] "Cancer genomics cloud." http://www.cancergenomicscloud.org/. [Online; accessed 2-February-2018].

[12] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *ACM SIGOPS operating systems review*, vol. 42, no. 5, pp. 64–74, 2008.

[13] J. Chicco, D. Collange, and A. Blanc, "Simulation study of new tcp variants," in *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pp. 50–55, IEEE, 2010.

[14] T. S. Gopal, R. Jain, P. R. L. Eswari, G. Jyostna, and S. R. Kamatham, "Securing udt protocol: Experiences in integrating transport layer security solutions with udt," in *Innovative Computing Technology (INTECH), 2013 Third International Conference on*, pp. 87–91, IEEE, 2013.

[15] "Aspera high-speed file transfer software." http://www.asperasoft.com. [Online; accessed 2-February-2018].

[16] M. R. Meiss *et al.*, "Tsunami: A high-speed rate-controlled protocol for file transfer," *Indiana University*, 2004.

[17] Y. Ren, H. Tang, J. Li, and H. Qian, "Performance comparison of udp-based protocols over fast long distance network," *Information technology journal*, vol. 8, no. 4, pp. 600–604, 2009.

[18] Y. Gu and R. L. Grossman, "Udt: Udp-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[19] J. Postel, "Transmission control protocol," 1981.

[20] J. Widmer, R. Denda, and M. Mauve, "A survey on tcp-friendly congestion control," *IEEE network*, vol. 15, no. 3, pp. 28–37, 2001.

[21] T. S. Gopal, N. Gupta, R. Jain, S. R. Kamatham, and P. R. L. Eswari, "Experiences in porting tcp application over udt and their performance analysis," in *Innovation and Technology in Education (MITE), 2013 IEEE International Conference in MOOC*, pp. 371–374, IEEE, 2013.

[22] L. Eggert and G. Fairhurst, "Unicast udp usage guidelines for application designers," tech. rep., 2008.

[23] R. L. Grossman, Y. Gu, X. Hong, A. Antony, J. Blom, F. Dijkstra, and C. de Laat, "Teraflows over gigabit wans with udt," *Future Generation Computer Systems*, vol. 21, no. 4, pp. 501–513, 2005.

[24] Y. Gu, X. Hong, and R. Grossman, "An analysis of aimd algorithms with decreasing increases," *Proceedings of GridNets' 04*, 2004.

[25] N. F. Pub, "197: Advanced encryption standard (aes)," *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[26] J. Jonsson, K. Moriarty, B. Kaliski, and A. Rusch, "Pkcs# 1: Rsa cryptography specifications version 2.2," 2016.

[27] "Orcid." https://orcid.org/. [Online; accessed 19-December-2017].

[28] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pp. 101–102, IEEE, 2001.

[29] X. Yue, W. Chen, and Y. Wang, "The research of firewall technology in computer network security," in *Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference on*, vol. 2, pp. 421–424, IEEE, 2009.

[30] A. Wacker, G. Schiele, S. Holzapfel, and T. Weis, "A nat traversal mechanism for peer-to-peer networks," in *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pp. 81–83, IEEE, 2008.

[31] J. Wagner and B. Stiller, "Udp hole punching in tomp2p for nat traversal,"

[32] Z. Hu, "Nat traversal techniques and peer-to-peer applications," in *HUT T-110.551 Seminar on Internetworking*, pp. 04–26, Citeseer, 2005.

[33] H. C. Phuoc, R. Hunt, and A. McKenzie, "Nat traversal techniques in peer-to-peer networks," in *Proceedings of the New Zealand Computer Science Research Student Conference (NZCSRSC)*, 2008.

[34] P. Matthews, R. Mahy, and J. Rosenberg, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," 2010.

[35] G. Peng, "Cdn: Content distribution network," *arXiv preprint cs/0411069*, 2004.

[36] "Digitalocean: Cloud computing, simplicity at scale." https://www.digitalocean.com/. [Online; accessed 12-April-2018].

[37] "Download test files - thinkbroadband." https://www.thinkbroadband.com/download. [Online; accessed 10-March-2017].

[38] "Tsunami udp protocol." http://tsunami-udp.sourceforge.net/. [Online; accessed 10-March-2018].

[39] "Github - johannesbuchner/udt-java: Java implementation of udt." https://github.com/JohannesBuchner/udt-java. [Online; accessed 10-March-2017].

[40] "Github - barchart/barchart-udt: Java wrapper for native c++ udt protocol.." https://github.com/barchart/barchart-udt. [Online; accessed 10-March-2017].

[41] "Github - sebsto/tsunami-udp: Fork of tsunami udp file transfer server and client w/fix to compile on os x mavericks." https://github.com/sebsto/tsunami-udp. [Online; accessed 10-January-2018].

[42] M. Ahmad, M. A. Ngadi, and M. M. Mohamad, "Experimental evaluation of tcp congestion contorl mechanisms in short and long distance networks.," *Journal of Theoretical & Applied Information Technology*, vol. 71, no. 2, 2015.

[43] "Github - java-native-access/jna: Java native access." https://github.com/java-native-access/jna. [Online; accessed 20-May-2018].

[44] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek, "The measured performance of content distribution networks," *Computer Communications*, vol. 24, no. 2, pp. 202–206, 2001.

[45] R. C. Merkle, "A certified digital signature," in *Conference on the Theory and Application of Cryptology*, pp. 218–238, Springer, 1989.