NAIST-IS-DD1561026

Doctoral Dissertation

SDN-enabled GridFTP: High speed data transfer system based on multiple TCP streams using OpenFlow

Huang Che

March 7, 2018

Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Huang Che

Thesis Committee:

Professor Hajimu Iida	(Supervisor)
Professor Kazutoshi Fujikawa	(Co-supervisor)
Associate Professor Kohei Ichikawa	(Co-supervisor)
Associate Professor Yasuhiro Watashiba	(Co-supervisor)

SDN-enabled GridFTP: High speed data transfer system based on multiple TCP streams using OpenFlow*

Huang Che

Abstract

A large amount of data needs to be transferred from one site to another as fast as possible in the fields of computational science. To achieve high-speed data transfer in widely-distributed environments, many applications utilize multiple TCP streams. Using multiple TCP streams in parallel can improve aggregate bandwidth by mitigating the negative effects of packet loss and the slow start mechanism of TCP. However, since multiple TCP streams of applications are usually routed according to the default IP routing protocol, only a single shortest path among the multiple paths can be utilized for the data transfer. This research proposes a multipath controller that increases the performance of data transfer by leveraging multiple paths simultaneously for parallel TCP streams.

For this purpose, we utilize the Software-Defined Networking (SDN) technology and its implementation, OpenFlow. Furthermore, we propose a prediction model to determine optimal numbers of parallel TCP streams to be assigned for each path according to its own network condition. This thesis presents the design and implementation of the proposed system. As a case study, we applied our proposed system on GridFTP and evaluated the performance improvement in both a virtual and a real global-scale environment. The results demonstrate that our proposed system accelerates the data transfer of GridFTP. In our real global-scale environment, our experimental results show the practicality of our proposal

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, NAIST-IS-DD1561026, March 7, 2018.

and indicate that our proposed method has achieved the performance close to the physical limitations of the hardware.

Keywords:

Software-Defined Networking, OpenFlow, GridFTP, Multipath, Data transfer

Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview

The rapid development of science, technology and Internet has led us into the big data era. Every industry produces a huge amount of data every day, such as Google and Facebook processing hundreds of petabytes of data per day [1]. The data of the world is still increasing exponentially due to advancing technology, such as the Internet of Things (IoT) [2], which is related to all aspects of our modern life. More than 90% of the data in the world today has been created in the last two years, and it has been estimated that we are generating 2.5 quintillion bytes of data per day [3, 4].

High-speed data transfer is a necessary technology for the process of big data [5]. The big data era gives us a lot of great opportunities and challenges in business and research. Researchers have made a lot of progress in developing the capability to compute, process, store, and analyze big data. In regards to data storage, most of the large-scale data is not stored only on one site but actually stored among the geographically distributed data centers around the world [6]. Therefore, high-speed data transfer between sites is a very important technology for international collaborative research and many other services.

To achieve high-speed data transfer in widely-distributed environments, many applications utilize multiple TCP streams simultaneously to transfer data. Using multiple TCP streams in parallel can improve aggregate bandwidth over using a single TCP stream by mitigating the negative effects of packet loss and 'slow start' mechanism of TCP. There have been a number of proposed schemes designed for applications to use multiple TCP streams such as XFTP [7], PSockets [8], GridFTP [9, 10], MultiTCP [11], PATTHEL [12], QTCP [13] Multipath TCP (MPTCP) [14], to increase the performance of data transfer.

On the other hand, there are usually multiple network paths (multipath) available between widely-distributed sites, however, these multiple paths are not efficiently utilized by applications. Because even if the applications use multiple TCP streams in parallel, those multiple TCP streams are basically routed according to the default IP routing protocol, and only a single shortest path among the multiple paths is used for the data transfer. The primary reason for this problem is that there is a gap between application demands and the network architecture, and the applications are unaware of the information on the network layer. Thus, there is still much room for improvement in data transfer by applying some traffic engineering technologies using different multiple paths simultaneously.

In this study, we propose a multipath controller that distributes the parallel TCP streams of applications into multiple network paths by utilizing Software-Defined Networking (SDN)-based traffic engineering techniques. SDN is a newly emerged concept that brings software programmability to networks and allows us to control routing assignment of the entire networks from the viewpoint of applications.

Furthermore, since each network path has its own network conditions, to fully utilize the network bandwidth, it is necessary to calculate the optimal number of TCP streams should be assigned for each network path. Therefore, to optimize the data transfer performance, we also developed a prediction model to determine the optimal numbers of parallel TCP streams to be assigned for each path according to its own network condition.

We have developed our system based on OpenFlow [15], which is standardized by the Open Networking Foundation (ONF) [16] and one of the most used standard protocols for SDN. We applied our proposed multipath controller to GridFTP as an actual case study to demonstrate the effectiveness and practicality of our proposed system, because GridFTP is one of the most common data transfer services using multiple TCP streams and it is used widely in the computational science research fields.

To verify the effectiveness and practicality of our proposed system, we performed evaluation experiments comparing the performance of the GridFTP with-/without our proposed method. For retrieving the best possible performance, we conducted the experiments in a virtual environment first. We then also performed

some experiments on a real global-scale environment to evaluate the practicality of our proposal.

1.2 Target and Use Cases

Our proposed system aims to provide a high-speed data transfer service to the computational science research projects. In every field of science, especially data-intensive scientific projects, large-scale data is produced and processed every day. However, most of these large-scale data is not stored only in one site, but actually stored across over the geographically distributed data centers around the world; thus, high-speed data transfer technique is very important for this data-intensive science.

For example, Compact Muon Solenoid (CMS) [17] and Laser Interferometer Gravitational-Wave Observatory (LIGO) [18] are well-known scientific projects in the physics research field. The purpose of CMS experiments is investigating a wide range of physics which built on the Large Hadron Collider (LHC) [19] at the European Organization for Nuclear Research (CERN). This project is supported by a global collaboration of more than 170 computing centers in 42 countries [20]. When the CMS experiment is conducted, more than 150 petabytes data is generated every time. These data need to be distributed to all sites around the world as soon as possible for the purpose of storing and analyzing. Obviously, these scientific projects consume significant storage and networking resources.

In addition, to support these scientific studies, most of the countries in the world provide the National Research and Education Network (NREN), such as the Energy Sciences Network (ESnet) [21], the pan-European research network (GEANT) [22], China Education and Research Network (CERNET) [23]. An NREN is a high-speed network resource which is essential in providing advanced Information and Communication Technology (ICT) services to the research and education communities. By utilizing the network resources of NRENs, large-scale data transfer is possible in these international collaborative researches. As a high-speed data technique, multiple TCP streams data transfer, such as GridFTP, is widely used by these scientific projects.

In this research, we focus on the use cases that integrate several NRENs and

create widely distributed shared computing infrastructures for the use of their scientific projects. In these kind of scientific projects, the number of participating organizations are relatively limited compare to the commercial Internet where the number of participating sites are unlimited. Therefore, the centralized control architecture of OpenFlow will work efficiently.

1.3 Organization of Thesis

The rest of this paper is organized as follows. Chapter 2 describes the background and related works, including various multiple TCP streams techniques, multipath routing methods, SDN and OpenFlow and existing researches on high-speed data transfer using multipath. Chapter 3 explains the implementation details of our proposed SDN-enabled GridFTP, including a multipath selection algorithm, a multipath OpenFlow controller and a new extended Globus XIO Driver. Chapter 4 describes the prediction model for our system. Chapter 5 explains the mechanism of proposed SDN-enabled GridFTP with prediction model in detail. Chapter 6 shows the evaluation results of the proposed system in both a virtual and real global-scale environment. Chapter 7 discusses possible issues when actually utilize our proposed system and our future work. Finally, Chapter 8 concludes this thesis.

2 Background

In this chapter, we will discuss the background of various techniques and research related to our work. Section 2.1 discusses the necessity of high-speed data transfer and describes various techniques for high-speed transfer. As a result of the discussion, we choose multiple TCP streams to realize high-speed data transfer. Then, we describe various techniques of multiple TCP streams and introduce the GridFTP in detail. Section 2.2 describes various techniques for multipath routing. Section 2.3 describes technique of Software-Defined Networking and OpenFlow in detail. Section 2.4 describes some related works and indicates features of our proposal.

2.1 Multiple TCP Streams for High-speed Data Transfer

High-speed data transfer service between sites is very important in the big data era. To meet the demands for high speed data transfer, the bandwidth of network has been improved continuously. In the network research field, 100 Gb/s end-to-end data communication network service (such as the Science Information Network 5(SINET5) of Japan [24]) is also provided by some organizations. However, it is still challenging to build a suitable system or a protocol to increase the utilization of network bandwidth.

Normally, many of data transfer applications rely on the Transmission Control Protocol (TCP) [25] for accurate and reliable data transmission. TCP is the dominant transport layer protocol for current IP networks and is used for more than 90% of total traffic [26–28]. However, since TCP provides flow control and congestion control functionalities [29, 30], it cannot fully utilize the network bandwidth.

To increase the utilization of the network bandwidth, three major approaches were developed by network researchers. The first approach is modifying standard TCP with congestion control algorithm and other parts. Many TCP variants have been developed [31–40]. The second approach is developing a new protocol using UDP [41] that can be also used from the application level [42–47]. The third one is using multiple concurrent TCP streams [7–9, 11–14].

Some proposed methods of the first and second approach can achieve an excellent utilization of network bandwidth as well as the third approach. For example, MulTCP [31] only uses one logical connection, but it can emulate like a set of multiple standard TCP connections to achieve high-speed transfer. However, according to several performance evaluations among these approaches [48,49], single stream of the TCP variants cannot overcome multiple simultaneous TCP streams especially in high Bandwidth-Delay Product (high-BDP) networks. Furthermore, to aggregate the bandwidth from available multiple paths, multiple TCP connections are necessary. Therefore, we have decided to focus on the use of multiple TCP streams in this research. In the following sections, we will discuss more detail about the technology using multiple TCP streams.

2.1.1 Multiple TCP Streams

Utilizing multiple TCP streams data transfer is an efficient way to achieve high-speed data transfer in widely-distributed environments. Using multiple TCP streams in parallel can improve aggregate bandwidth over using a single TCP stream by mitigating the negative effects of packet loss and slow start mechanism of TCP. There have been a number of proposed schemes designed for applications to use multiple TCP streams. We explain these schemes from different layers, including transport layer, session layer, and application layer.

2.1.2 Multiple TCP Streams in Transport Layer

There are a lot of protocols design at transport layer for utilizing multiple TCP streams, such as [50–54]. We explain several representative protocols. The Stream Control Transmission Protocol (SCTP) [50] is a unicast protocol and supports data exchange between exactly two endpoints. SCTP allows user's messages to

be delivered by multiple streams, but it is not clear how it can achieve the desired throughput in a congestion scenario.

Multipath Transmission Control Protocol(MPTCP) [14] is an extension of the TCP/IP stack that has been widely researched. When an application utilizes MPTCP, only a logical TCP was used by the application. Then MPTCP split the data from application across multiple subflows, each of which is a conventional TCP connection. It also provides a congestion control mechanism which takes care that traffic on a congested path is moved to a link with less congestion. Hence it adapts the load balancing according to the load of other traffic on the paths.

2.1.3 Multiple TCP Streams in Session Layer

PATTHEL [12] is a session layer protocol that achieves parallelization by creating multiple TCP channels between hosts. The protocol uses a dedicated channel created in first for control channel connection, the rest data channels are used to transfer data. A received data block from the application layer is divided into chunks of variable size depending on the channel characteristics. It also provides an Application Programming Interface (API) for the application developers to use this protocol. However, the performance of PATTHEL relies on a policy that supervises the opening and the closing of new channels. It includes a set of parameters that need to be fine-tuned on a case-by-case basis to achieve optimal performance.

2.1.4 Multiple TCP Streams in Application Layer

A lot of application-level implementations [7–11] have been proposed for utilizing multiple TCP streams in application layer. PSockets [8] is a library which helps wide area applications that need to move large amounts of data. It has the same API as that of regular sockets. As a use case, it was used by geographically distributed data-intensive computing application which was designed for high energy physics data transfer. The evaluation results have shown that the performance of data transfer was increased obviously.

Other implementations are similar to PSockets which are used in many high-

speed data scenarios. Application layer implementation is easy to make users deploy and utilize. And unlike transport layer implementation, it is not necessary to modify the kernel. In particular, the GridFTP has been commonly used especially in the field of the computational science research. Therefore, we will explain GridFTP in detail in next section and utilize GridFTP as the first option to realize multiple TCP streams in our research.

2.1.5 GridFTP

GridFTP [9, 10] is an extended data transfer protocol of the File Transfer Protocol (FTP) [55–57], that has been widely used in the Grid computing, and was standardized by the Global Grid Forum [58]. Normally, GridFTP uses TCP as its transport layer communication protocol and able to solve several problems of TCP. For example, besides the features of the existing FTP, GridFTP has additional useful features such as authentication, data integrity, data confidentiality, automatic negotiation of TCP buffer/window sizes, striped data transfer, parallel data transfer, third-party control of file transfer, partial file transfer, security, and reliable data transfer [59].

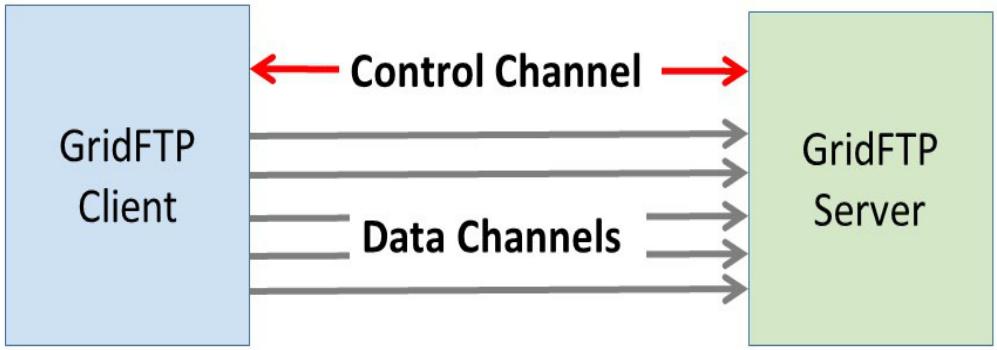

Figure 2.1: Parallel data transfer in Globus GridFTP

Most of these specific features of GridFTP are implemented by Globus [9]. Currently, GridFTP server and client software conforming to GridFTP is included in the Globus Toolkit [60], which is the de facto standard middleware for Grid

computing developed and provided by the Globus Alliance. Figure 2.1 shows parallel data transfer in Globus GridFTP in case of communication between a client and a sever (Note here that GridFTP also supports data transfer between a client and multiple servers simultaneously). As we can see, GridFTP has two channel protocol: control channel and data channel.

When GridFTP client wants to start data transfer, it first opens a control channel connection to the server side. The control channel is used to specify a put or get operation and encrypted by default. Then, it will individually open the number of data channel streams specified by the user. The data channel is responsible for actual data transfer.

GridFTP with UDT

UDP-based Data Transfer Protocol (UDT) [47, 61] is a high-performance data transfer protocol designed for transferring large volume datasets over high-speed wide area networks. UDT utilizes UDP to transfer large data with its own reliable control and congestion control mechanisms. This newly designed protocol can transfer data at a much higher speed than TCP [47].

We mentioned above, though GridFTP utilizes TCP by default, the architecture of GridFTP can easily support various transport layer communication protocol. In the latest release of Globus Toolkit (Version 6.0), GridFTP becomes more wildly supporting UDT. However, several performance evaluations [62, 63] show the limitations of UDT. The UDT seems to give better performance on network paths with small latency. Furthermore, in regard to system resources usage, UDT consumes much more resources when achieving almost same performance with TCP (The CPU utilization for TCP transfers was in the range of 30-50%, whereas for UDT transfers it was around 80%. The memory consumption was around 0.2% for TCP and 1% for UDT). Therefore, we utilize standardized TCP protocol as the transfer protocol for GridFTP in this research.

Limitations of GridFTP

Like other multiple TCP streams implementation such as MPTCP, even GridFTP can generate parallel TCP streams, it could only utilize a network which provided

by default IP routing. In other words, without additional network traffic engineering techniques, the maximum performance of GridFTP is limited by allocated network path.

In addition, the number of TCP streams of GridFTP is also an important factor which affects its performance. If the number of TCP streams is too small, GridFTP could not fully utilize network resources. In the contrary, if the number of TCP streams is too large which may cause performance degradation as following situations: (1) It would cause network congestion, then the window size per TCP stream becomes smaller, so TCP timeout would frequently occur. (2) The overhead of processing TCP protocol stack would increase. Therefore, the optimal number of TCP connections should be determined based on the network conditions. Nevertheless, how to optimize the number of parallel TCP connections has not sufficiently been studied and still remains as an open issue. Therefore, it is necessary to combine a network engineering technique to make GridFTP more efficient. In the next section, we will discuss the routing techniques for multipath.

2.2 Multipath Routing

Basically, multiple network paths exist between widely-distributed sites. However, due to the conventional design of the network model, only one of the paths is usually available for a communication. We discussed the benefit of multiple TCP streams in the previous section. However even if the applications use multiple TCP streams in parallel, those multiple TCP streams are basically routed according to the default IP routing protocol (such as BGP [64], RIP [65] and OSPF [66,67]), and only a single shortest path among the multiple paths is used for the data transfer. The primary reason for this problem is that the applications are unaware of the information on the network layer. Therefore, there is still much room for improvement in data transfer by applying multipath routing technologies using different multiple paths simultaneously.

Multipath routing is a routing technique that aims to provide multiple alternative paths by utilizing the underlying physical network. This technique can achieve many benefits for many network functions, such as load balancing, fault tolerance, and higher bandwidth utilization. To realize multipath routing, the

approaches of the implementing can be divided into three major designs including source or hop-by-hop routing, centralized or distributed routing and static or dynamic routing. We divide the multipath routing methods into source and hop-by-hop routing to explain from the perspective of route computation.

2.2.1 Source Multipath Routing

In source routing (path addressing), all the routing information from source to destination is first collected at the source side, then the source side can partially or completely specify the route to the packets which towards the destination side. Since source routing can meet the requirements of different applications, it is a good way to achieve optimization of path routing. The key point of this routing mechanism is discovering the routes between sites through exchanging control messages, and many studies [68–75] have been devoted to source multipath routing approach. We explain several typical studies as follows.

To support a more flexible traffic engineering in IP-based networks, the multiprotocol label switching (MPLS) [76] where IP packets are switched through the pre-established Label Switched Paths (LSPs) by signaling protocols and has widely deployed in internet service providers (ISPs). For example, Seok et al. [77] proposed a dynamic multipath traffic engineering scheme which utilizes source routing and MPLS traffic engineering with a constraint on the total number of hops and paths.

In addition, the method to realize global-scale source multipath routing is a big challenge [78]. Internet separates routing into intradomain routing and interdomain routing. The intradomain routing focuses on optimal routing within a single autonomous system (AS), while the latter focuses on ensuring routing between multiple ASs. The Border Gateway Protocol (BGP) [64] is the de facto interdomain routing protocol of the global-scale Internet. However, most routing mechanisms based on BGP are not optimal in terms of cost, performance or reliability. Therefore, many studies such as BANANAS [69], new Internet routing architecture (NIRA) [72] diverse new mechanisms or techniques to adapt BGP protocol and make the adjustment possible in AS-level, which can make source side able to utilize multipath.

2.2.2 Hop-by-Hop Multipath Routing

In hop-by-hop routing, the source side cannot make a decision on packets to go through expected network path. The forwarding decisions are made by each node (or network device) which connects source and destination. Each node forwards a packet to a specified link based on the destination address of incoming packet header and its corresponding longest prefix match entry in the forwarding table stored in memory. Hop-by-hop routing is the most famous and widely used technique in IP networks.

To realize multipath by using hop-by-hop routing. Equal Cost Multipath (ECMP) [66, 79] is a multipath routing technique that has been adopted in many routing protocols like OSPF and Intermediate System to Intermediate System (ISIS) [80]. A network device with ECMP distributes the load equally over multiple equal-cost paths and utilizes simple round-robin method. Since ECMP is designed for equal-cost networks and will not exhibit multipathing when path costs are not equal. Some studies [67, 81] have extended ECMP and make it available over unequal paths.

There are also some extended version of BGP routing proposed to realize hop-by-hop multipath routing [82–85]. For example, R-BGP [84] applied multipath routing to hierarchical networks including different ASs. Besides the primary multiple paths, there are also some additional paths served as backup paths, which aims to maintain connectivity even in case of multiple paths failure. Yet another multipath routing (YAMR) [85] is another BGP extension. It can be used to construct multiple paths and establish concurrent transmission in hierarchical networks. YAMP also provides reliable communication against any single interdomain link failure and it is able to reduce control overhead by isolating failures.

2.2.3 Problems of Current Multipath Routing

All the current multipath routing as we shown are utilized additional header/label or extended routing protocols. However, these multipath routing methods would cause additional overhead in both the control and data planes.

In control plane, exchanging the extra topology or path information required

for multipath routing would consume extra network bandwidth and processing resources. Computing multiple paths would require more computational power on each node. In data plane, forwarding traffic on different paths requires the data packets to carry an extra header or label and this would consume more memory.

Furthermore, these methods are not easy to implement and adapted to the entire Internet. Without entire control over the end-to-end network, it is not possible to provide an appropriate multipath routing. Due to limitations of the current network, Software-Defined Networking (SDN) technique which separates control plane and data plane are necessary for the future network. Therefore, we can utilize SDN to provide feasible multiple routing mechanisms and distributes the multiple TCP streams of the application into multiple network paths.

2.3 Software-Defined Networking and OpenFlow

2.3.1 Software-Defined Networking

Software-Defined Networking (SDN) is an emerging concept that is dynamic, manageable, cost-effective, and adaptable for the future network. SDN is expected to meet for the high-bandwidth, dynamic nature of today's applications. By separating the network control and forwarding functions, SDN makes the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services.

In conventional networking, data plane and control plane are implemented in the firmware of network equipment, such as routers and switches. Data plane processes the movement of actual data which entering the network equipment. Control plane deals with routing decision such as making a decision on where to send frames or packets. Those control planes which are implemented in the different networking devices, cooperate with each other and decide the behavior of the entire network.

Figure 2.2 illustrates the SDN architecture, which consists of three layers. The lowest layer is the infrastructure layer, also called the data plane. Data plane comprises the forwarding network equipment. The responsibilities of the data

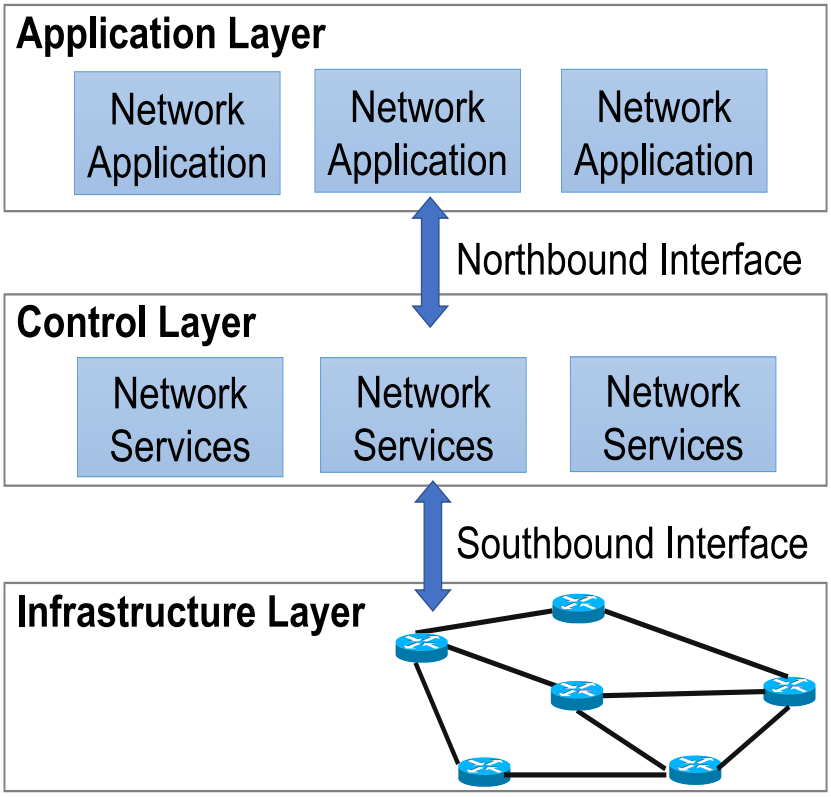

Figure 2.2: A three-layer Software-Defined Networking architecture.

plane are mainly forwarding data, as well as monitoring local information and gathering statistics.

One layer above is the control layer, also called the control plane. It is responsible for programming and managing the data plane. Control plane uses the information provided by the data plane and defines network operation and routing. It comprises one or more software controllers that communicate with the forwarding network elements through standardized interfaces, which are referred to as southbound interfaces. We describe OpenFlow, a protocol option for the southbound interface in section 2.3.2.

The top layer is application layer. This layer contains network applications that can introduce new network features, such as security and manageability, forwarding schemes or assist the control layer in the network configuration. The application layer can receive an abstracted and global view of the network from

the controllers and use that information to provide appropriate guidance to the control layer. The interface between the application layer and the control layer is referred to as the northbound interface.

2.3.2 OpenFlow

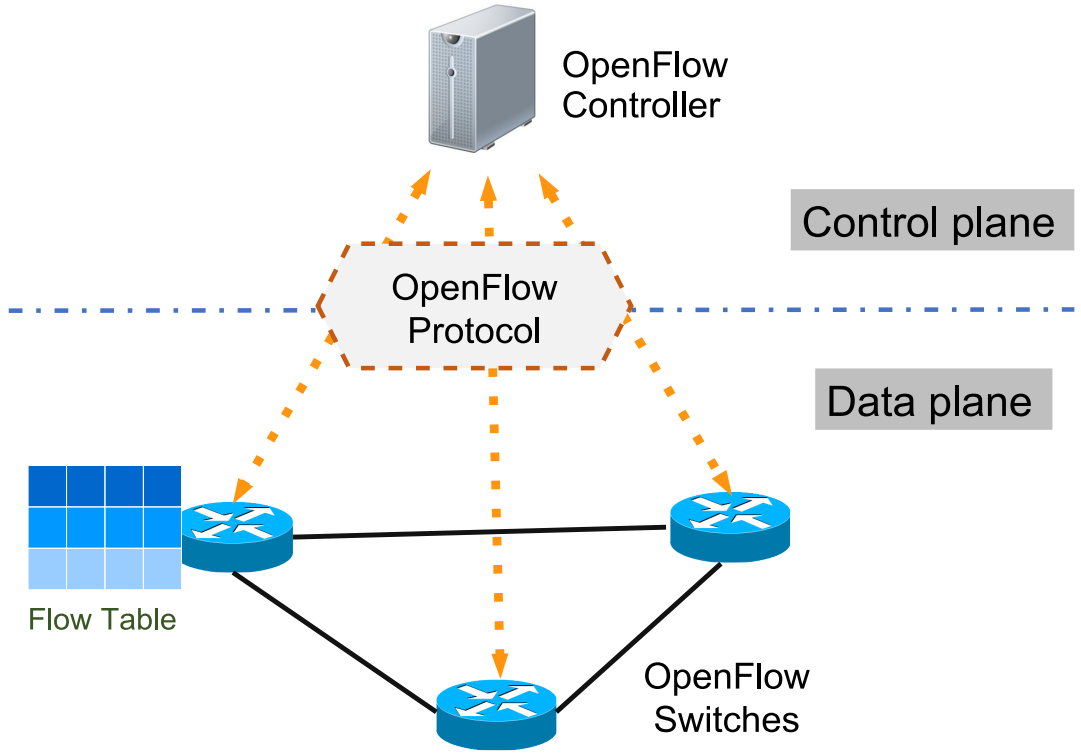

Figure 2.3: Overview of an OpenFlow network.

To realize the SDN concept, one of the most used standard protocols for southbound interfaces is OpenFlow, which is standardized by the Open Networking Foundation (ONF). Figure 2.3 shows the overview of an OpenFlow network. The OpenFlow network separates the data plane from the control plane and consists of three basic concepts.

The first concept is the OpenFlow network is connected with OpenFlow switches that compose the data plane. An OpenFlow switch is a basic data forwarding network equipment that forwards packets according to its own flow table. This

table holds a set of flow entries, each of which consists of match fields, counters, and instructions. Flow entries are also called flow rules.

Second concept is the control plane consists of one or more OpenFlow controllers. The OpenFlow specification defines the protocol that enables the controller to instruct the switches. By insertion, modification and removal of flow entries inside OpenFlow switches, the OpenFlow controller is able to modify and influence the network. The part of OpenFlow controller is programmable. There are also a lot of frameworks for developing OpenFlow controller, such as Trema [86], POX [87], NOX [88], Floodlight [89], BEACON [90], OpenDaylight [91], Ryu [92].

Lastly, the communication procedure between the data plane and the control plane is specified by OpenFlow protocol. In particular, OpenFlow controller and OpenFlow switches communicate with each other through a secure control channel.

As shown above, OpenFlow can provide a flexible programming network and make the network easy to manage. Hence, unlike traditional TCP/IP protocol, OpenFlow is the good option that provides flexible traffic engineering. We can utilize OpenFlow to provide multipath and distribute the multiple TCP streams of GridFTP into the different paths to aggregate more bandwidth. Therefore, we will combine SDN with GridFTP to propose a high-speed data transfer system.

2.4 Related Work

There have been a number of proposed schemes for providing multipath to applications [93–95]. Some of them have the similar approach to our study. We describe several examples as follows.

Kissel et al. developed a new session layer protocol, called Phoebus, and made it available from the applications by taking advantage of the Dynamic Circuit Network (DCN), which is deployed on national academic research networks such as Internet2 [96]. Phoebus provides Phoebus Gateways (PGs), which hide different transport layer protocols such as UDT [61] and TCP behind the session layer, and offer an easy method to improve the throughput for general applications.

In addition, Dan et al. extended the research idea of Kissel et al. and pro-

posed a system that improves the network throughput by assigning multiple TCP streams to different multiple paths brought on the conventional IP-based routing, the DCN provided by Phoebus and the OS^{3}E of Network Development and Deployment Initiative (NDDI) [97]. NDDI/OS^{3}E is a service that builds Layer 2 networks dynamically on their OpenFlow network infrastructure.

The research of Kissel et al. leveraged high-speed circuits brought by the DCN of Internet2 and made the multiple transport protocols available on the DCN. However, their method does not provide an interface for the users to control routing paths inside the Internet2. Once, a path is provided through the DCN, the provided path is just statically used for the multiple TCP streams of application. Dan et al. use a Layer 2 network dynamically built upon their OpenFlow network provided by NDDI/OS^{3}E. However, the users cannot control each individual routing inside the NDDI/OS^{3}E. The routes for the Layer 2 network is determined at the time it is created on the OpenFlow network. Therefore, Dan et al.'s system also just assigns the multiple routes provided by NDDI/OS^{3}E to the multiple TCP streams of application. Therefore, those two approaches are not flexible enough to optimize multiple paths between widely distributed sites.

However, recently, the optimization of the network layer based on the requests from the application layer has been gathering much attention with the development of the OpenFlow network. The Research Infrastructure for large-scale network Experiments (RISE) service provided by Japan Gigabit Network eXtreme (JGN-X) is one of the services that allow users to control individual OpenFlow switches of the service [98, 99]. Considering the recent trend of the research, it is necessary to optimize multipath assignment by controlling the individual switches under an environment where OpenFlow switches are available for end-to-end communication.

Although the above existing researches tried to generate multiple TCP streams at the application level, a method generating multiple TCP streams at the system level, Multipath TCP (MPTCP), has also been proposed recently. There is also a research [54] that combines MPTCP and OpenFlow to achieve high-speed data transfer. However, the research also did not consider an environment where OpenFlow switches are available for end-to-end communication. In terms of routing multiple TCP streams, the difference between multiple streams of application

level and multiple streams of MPTCP does not matter essentially. In this study, we evaluate our proposed method with application-level multiple streams using GridFTP, because GridFTP has already been widely used for multiple TCP streams while MPTCP is not available widely.

3 SDN-enabled GridFTP

To achieve further high performance by using multiple TCP streams, it is necessary to provide available multipath. As we discussed in the session 2.3, using OpenFlow technique is a possible solution to realize multipathing routing. Therefore, we propose the SDN-enabled GridFTP that achieves high-speed data transfer by assigning multiple TCP streams of GridFTP to different paths in the environment where OpenFlow network are available for the communication. By aggregating the available bandwidth from multiple different paths, SDN-enabled GridFTP would improve the performance of data transfer drastically.

In this chapter, we explain the approach and design of our high-speed transfer system. We also describe the implementation details of our proposed SDN-enabled GridFTP system, including a multipath selection algorithm, a multipath OpenFlow controller and a new extended Globus XIO Driver.

3.1 Approach and Design

In this study, we propose a system that tries to improve the data transfer performance of GridFTP by assigning parallel TCP streams of GridFTP to a number of different paths in the environment where OpenFlow switches are available for end-to-end communication. By aggregating the available bandwidth from multiple different paths, the performance of data transfer would be improved drastically.

Usually, there are several network paths between different sites. However, one of the shortest paths is used solely for data transfer in the default IP routing. Figure 3.1 shows the parallel transfer of the conventional GridFTP. GridFTP tries to increase the data transfer performance by creating multiple TCP streams on a single network path and by fully utilizing the bandwidth of the path. However,

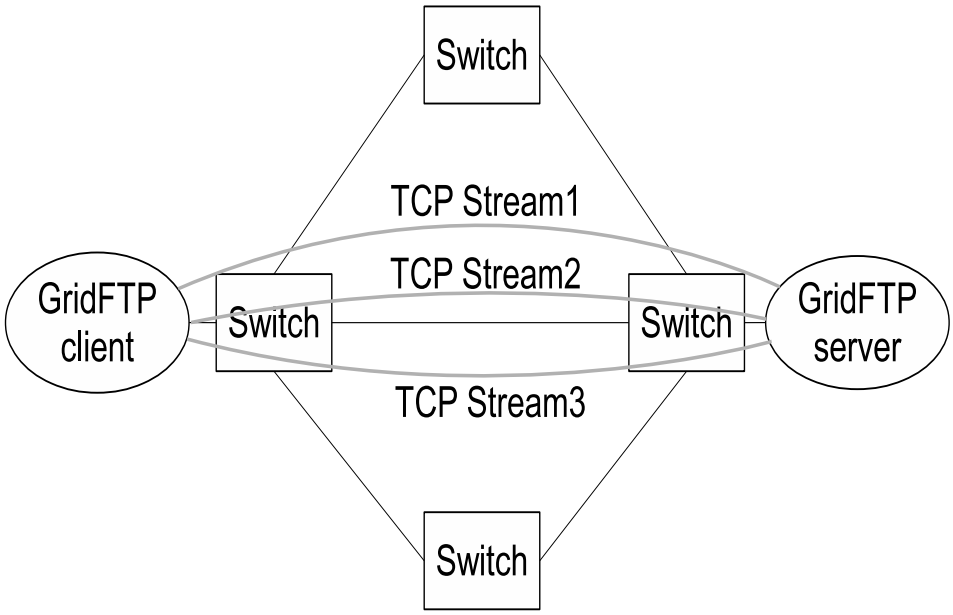

Figure 3.1: Parallel transfer of the conventional GridFTP

the line speed is limited to the selected network path. Therefore, the maximum bandwidth is limited even if GridFTP uses multiple TCP streams on the selected path.

Figure 3.2 shows our proposed parallel transfer of GridFTP in an OpenFlow network. We try to aggregate multiple routes to increase the available bandwidth by assigning each of multiple TCP streams to different routes. In this study, we construct an OpenFlow controller that dynamically calculates available paths using breadth-first search between sites based on the request from the applications and allocates the calculated routes for the applications.

The purpose of this study is to improve the data transfer performance by using GridFTP. However, we carefully designed the system not to depend on GridFTP. We have designed general interfaces to request multiple routes so that any application can request multiple routes for data transfer. The following two functionalities are designed for the purpose: 1) Searching the specified number of paths between sites and holding the found paths for later use; 2) Installing appropriate flow entries into OpenFlow switches in response to the request of TCP connection from applications.

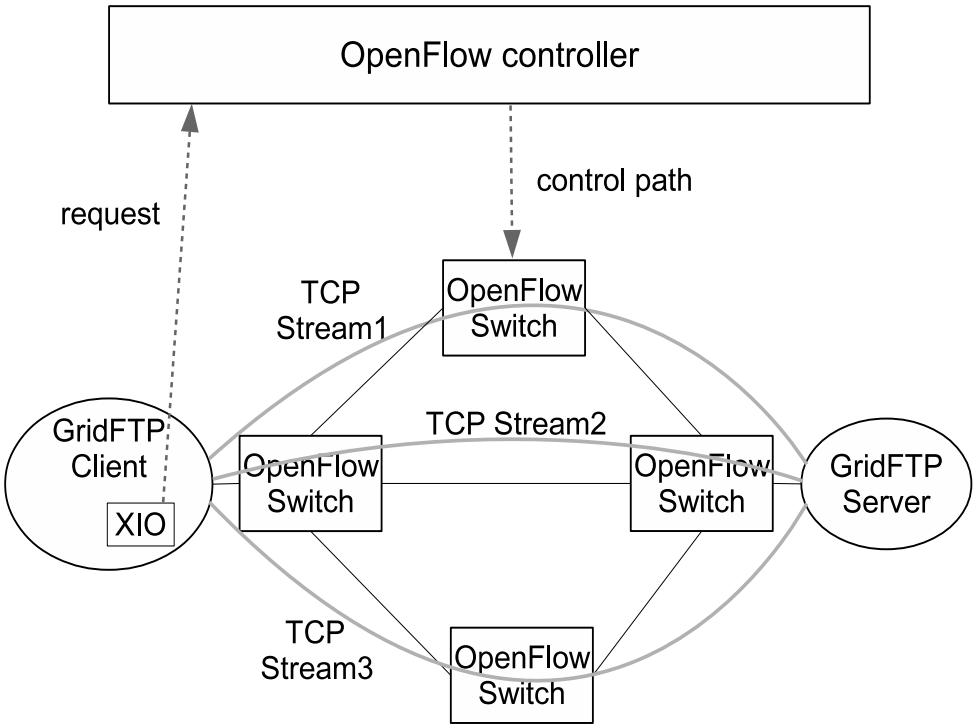

Figure 3.2: Parallel transfer of our proposed GridFTP over OpenFlow network

The reason why we have two separate functions to assign multiple paths is that the TCP port numbers used for a communication are not determined until just before opening the TCP connection. Since the multiple TCP streams of GridFTP have the same source and destination IP addresses, we utilize the port number of each TCP stream to distinguish and distribute multiple TCP streams. To install a flow entry into an OpenFlow switch, we need information on a set of source and destination addresses and TCP port numbers. Therefore, we search multiple available paths between sites first when the source and the destination addresses are provided. Then, we create actual flow entries when TCP port numbers are determined. Thus, we designed two separate functions to find multiple available routes and assign the routes for each actual TCP connection.

3.2 Implementation

To meet the functionalities of our proposed SDN-enabled GridFTP system as we described in the previous section, we develop our proposed SDN-enabled GridFTP system by implementing a multipath selection algorithm, a multipath OpenFlow controller and a new extended Globus XIO Driver.

3.2.1 Multipath Selection Algorithm

Algorithm 1 Algorithm for calculating N paths from A to B

Require: $A, B \in Switches; N \geq 0$

1: $G(V, E) \Leftarrow NetworkTopology(Switches, Links)$
2: srating from vertex A
3: create a queue Q
4: mark A as visited and put A into Q
5: **while** Q is non-empty **do**
6: remove the head v of Q
7: mark and enqueue all unvisited node
8: **if** B is neighbours of v **then**
9: path is found
10: $AllPaths \Leftarrow path$
11: **end if**
12: **end while**
13: **if** $0 < N < length(AllPath)$ **then**
14: $SelectedPaths = N$ paths from $AllPaths$ in order
15: **return** $SelectedPaths$
16: **else**
17: $SelectedPaths = AllPaths$
18: **return** $SelectedPaths$
19: **end if**

Whether or not the controller can provide an appropriate multipath set for the GridFTP directly affects the data transfer performance of GridFTP. There are many algorithms to choose multiple paths from source to destination. Since

we will use a prediction model (explained in section 4) to calculate the optimal number of TCP streams for each path and optimize multiple TCP streams according to network condition of each path, we can fully use the assigned network resources. Therefore, we do not focus so much on optimizing the path selection, and utilize a simple searching algorithm, breadth-first search algorithm, to calculate available paths.

Algorithm 1 describes the method to find multipath for our system. Our multiple OpenFlow controller gathers the connection information from OpenFlow switches which connect to the controller. By utilizing the gathered information, the controller can make a network topology graph. When a request for N paths is made to the controller, the controller first utilizes breadth-first search algorithm to calculate all available paths between the source and destination. Then the controller will provide paths according to the number N specified by the users. If N is less than the number of total calculated paths, the controller will return the N paths from all paths in order. Otherwise, all paths will be returned by the controller. If the user specify 0 as N, the controller also returns all path set.

3.2.2 Multipath OpenFlow Controller

There are a lot of frameworks for developing OpenFlow controller as shown in section 2.3.2. We developed our multipath controller using Trema (version 0.4.6) [86], a framework for developing OpenFlow controllers in Ruby and C. We developed our controller based on the *routing_switch controller* [100] included in Trema Apps [101]. Trema Apps is a sample application set for Trema. *Routing_switch controller* is a simple OpenFlow controller that calculates the shortest-hop path between hosts using Dijkstra's algorithm and installs flow entries into the OpenFlow switches for the path. Our controller utilizes this default shortest-hop routing for the normal communications that do not request multipath routing.

We have implemented the two functionalities mentioned in the previous section, 1) Searching the specified number of paths between sites, and 2) Installing flow entries into OpenFlow switches, on our OpenFlow controller. In order to realize these two functionalities, we implemented two interfaces called AssignMultipath (for functionality 1) and MakePath (for functionality 2).

In addition, as mentioned previously, we have designed the controller so that

any other applications than GridFTP can utilize the controller. For the purpose we implemented the two functionalities as common XML remote procedure call (XML-RPC) [102] interfaces. XML-RPC is designed as a simple protocol allows software running among heterogeneous operating systems or different environments to make procedure calls across the network. It utilizes HTTP as the transport of remote procedure calling and XML as the encoding. XML-RPC implementations can be available under any kind of operating systems, programming languages, dynamic and static environments. In this study, we utilized C++ implementation of the XML-RPC (version 0.7) to realize the interface of our controller.

Listing 3.1 describes the XML interface for the request to the *AssignMultipath*. The request includes three parameters, `src_ip`, `dst_ip` and `path_num`. The `src_ip` indicates the IP address of source host. `dst_ip` specifies the IP address of destination host. The `path_num` is the number of paths which user want to utilize for data transfer. As mentioned in the previous section, *AssignMultipath* calculates the available multipath using Algorithm 1 and also calculates the optimal number of TCP streams to be used by using the proposed prediction model of optimal TCP stream assignment.

Listing 3.2 describe the XML interface for the response from the *AssignMultipath*. *AssignMultipath* responses two values, `path_set_id` and `tcp_num`. The `path_set_id` is a unique ID identifying the assigned paths set which will be used later for calling the *MakePath* interface. The `tcp_num` specifies the predicted optimal number of multiple TCP streams that should be used by the application.

Listing 3.3 describes the XML interface for the request to the *MakePath*. This request was made by the application when it opens a new TCP connection. The request includes two parameters, `path_set_id` and `tcp_port_num`. The `path_set_id` specifies the unique ID identifying the path set assigned by AssignMultipath. The `tcp_port_num` is the source TCP port number of each TCP stream which is used by the application. *MakePath* retrieves an available path from the specified path set and install flow entries on the OpenFlow switches along the path accordingly. When *MakePath* has successfully installed the flow entries it just closes the connection of the XML-RPC call without any response.

```
<?xml version="1.0"?>
<methodCall>
 <methodName>AssignMultipath</methodName>
 <params>
  <param><value>
   <struct>
    <member>
     <name>src_ip</name>
     <value>
      <string> source IP address </string>
     </value>
    </member>
    <member>
     <name>dst_ip</name>
     <value>
      <string> destination IP address </string>
     </value>
    </member>
    <member>
     <name>path_num</name>
     <value>
      <int> path number specified by user </int>
     </value>
    </member>
   </struct>
  </value></param>
 </params>
<methodCall>
```

Listing 3.1: The XML-RPC request to AssignMultipath

```
<?xml version="1.0"?>
<methodResponse>
 <params>
 <param><value>
  <struct>
   <member>
    <name>path_set_id</name>
    <value><int>
     A unique ID identifying the assigned paths
    </int></value>
   </member>
   <member>
    <name>tcp_num</name>
    <value><int>
     the number of multiple TCP streams
     that should be used by GridFTP
    </int></value>
   </member>
  </struct>
 </value></param>
 </params>
</methodResponse>
```

Listing 3.2: The XML-RPC response from AssignMultipath

```
<?xml version="1.0"?>
<methodCall>
 <methodName>MakePath</methodName>
 <params>
 <param><value>
  <struct>
   <member>
    <name>path_set_id</name>
    <value><int>
     A unique ID identifying the path set
     assigned by AssignMultipath
    </int></value>
   </member>
   <member>
    <name>tcp_port_num</name>
    <value><int>
     The port number of each TCP stream
    </int></value>
   </member>
  </struct>
 </value></param>
 </params>
</methodCall>
```

Listing 3.3: The XML-RPC request to MakePath

3.2.3 Globus XIO Driver for SDN-enabled GridFTP

In this study, we tried not to modify the design of the current GridFTP too much, and we carefully minimized the modifications, so that we can easily and widely apply the proposed implementation to the existing GridFTP deployment. For the purpose, we implemented the required functionality to support our proposed system as one of the Globus XIO [9] communication drivers that GridFTP uses as a communication library. Since the implementation of GridFTP is separated from that of the Globus XIO libraries, the modification of XIO libraries does not affect the implementation of the GridFTP. Globus XIO is a plug-in framework of the I/O libraries that are implemented in Globus Toolkit [60]. In our implementation, we utilized the Globus Toolkit 5.2.5 which is a stable version and compatible with our environment.

Globus XIO allows applications to support various protocols and file formats by implementing plug-ins for each different communication method and file and storage access method. Figure 3.3 illustrates the Globus XIO architecture. Globus XIO is comprised of two main components: framework and driver stack. The Globus XIO framework manages I/O operation requests that an application makes via the user API. The framework does not manipulate or deliver the data in an I/O operation; the drivers do all of that work. The task of the framework is to manage requests and map them to the interface of drivers.

Driver stack of Globus XIO has many types of drivers. The drivers are responsible for manipulating and transporting the data from the user. There are two types of drivers: transform and transport. Transform drivers are responsible for manipulating the data buffers passed to it via the user API and the XIO framework. Transport drivers are responsible for sending the data over a wire. When an I/O operation is requested, the Globus XIO framework passes the operation request to every driver in the order the drivers are in the stack. When the bottom-level driver (the transport driver) finishes shipping the data, it passes the request completion notification back to the XIO framework. Globus XIO then delivers the notification back up the stack in this manner until it reaches the top, at which point the application is notified that its request is completed.

In addition, there must be only one transport driver in a driver stack, and the transport driver must be at the bottom of the stack. The reason is that the

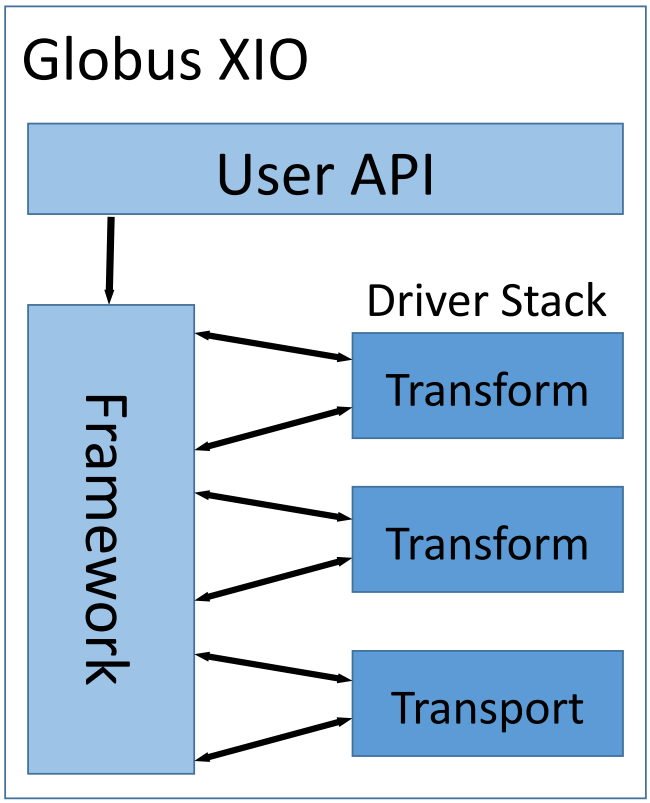

Figure 3.3: Globus XIO architecture

transport driver is responsible for the actual data move on a network path. The protocols that use multiple transport drivers have to construct multiple stacks. For example, a protocol that uses TCP for exchanging control information and UDP for transferring the actual data needs two different stacks: one with TCP as a transport and the other with UDP as the transport driver. Any number of transform drivers can be in a stack. Since we utilize GridFTP with only TCP protocol, we just need to provide one driver stack including a new transport driver.

In this study, we created a new communication driver based on the standard XIO TCP driver which is one of built-in default drivers in Globus XIO. We implemented our communication driver establishes TCP connections in collaboration with our multipath OpenFlow controller so that each connection can take different paths. As we mentioned in the previous section, multipath OpenFlow

controller needs information on a set of source and destination addresses and TCP port numbers to install flow entries to OpenFlow switches. Furthermore, since the source and destination addresses of multiple TCP streams of GridFTP are same, we have to provide the source TCP port number to OpenFlow controller in order to distinguish the different TCP streams. For the purpose, we modified the following function in the default XIO TCP driver.

```
globus_xio_tcp_driver.c:

globus_l_xio_tcp_bind(
    globus_xio_system_socket_t          fd,
    const struct sockaddr *             addr,
    int                                 addr_len,
    int                                 min_port,
    int                                 max_port,
    globus_bool_t                       listener)
```

This function is used to bind a specific port number to a newly opened TCP socket. Therefore, we added the following codes so that it tells the assigned source TCP port to the controller by calling the XML-RPC function.

```
if(!first_local_bind) {
    char *MPATH_ASSIGNMENT_ID;
    MPATH_ASSIGNMENT_ID = getenv("MPATH_ASSIGNMENT_ID");
    makepath(atoi(MPATH_ASSIGNMENT_ID),port);
}
```

Since GridFTP opens a TCP based control connection before data transfer. We make the control connection utilize the default shortest path provided by the OpenFlow controller. Hence, the second and later source port number of TCP streams will be used in our implementation. The `MPATH_ASSIGNMENT_ID` is an environment variable which includes the unique ID of the path set assigned by *AssignMultipath* of OpenFlow controller. The XML-RPC client will send a request to the OpenFlow controller with the source TCP port number and the unique ID.

In this way, when GridFTP tries to open a new TCP stream for data transfer, the proposed new XIO driver informs the source TCP port number to the multipath OpenFlow controller behind the call. This approach allows the GridFTP to support our OpenFlow network without any modifications on the GridFTP itself. Also, other applications using Globus XIO may benefit from the proposed system.

4 Prediction Model to Optimize TCP Stream Assignment in Multipath Routing

In section 3, we proposed a multipath controller that improves the data transfer performance by assigning parallel TCP streams of GridFTP to a number of different paths in the environment where OpenFlow switches are available for end-to-end communication. By aggregating the available bandwidth from multiple different paths, the performance of data transfer would be improved drastically.

However, to achieve the best performance using multiple paths, the strategy of how many TCP streams should be assigned for each path is also an important factor. The simplest method is to create enough TCP streams and distribute these TCP streams equally over the multiple paths. However, this is obviously inefficient in terms of resource usage. Therefore, it is necessary to figure out the optimal combination of multipath and the number of parallel TCP streams.

We therefore tried to develop a method to determine optimal numbers of parallel TCP streams to be assigned for each path according to its own network condition. There are many factors affecting the transfer speed in a network connection; the two major factors are the bandwidth and the latency. Since each network path has different bandwidth and latency, the optimal number of parallel TCP streams to get the best performance may be different for each path. In order to develop a prediction model to determine optimal numbers of TCP streams, we figured out the relationship between the optimal number of TCP streams and network conditions.

4.1 Goal of the Prediction Model

Our purpose of developing a prediction model is to calculate the optimal number of TCP streams for each network path according to its own network condition. Multiple TCP streams can increase the utilization of network bandwidth by aggregating the performance that each TCP achieved.

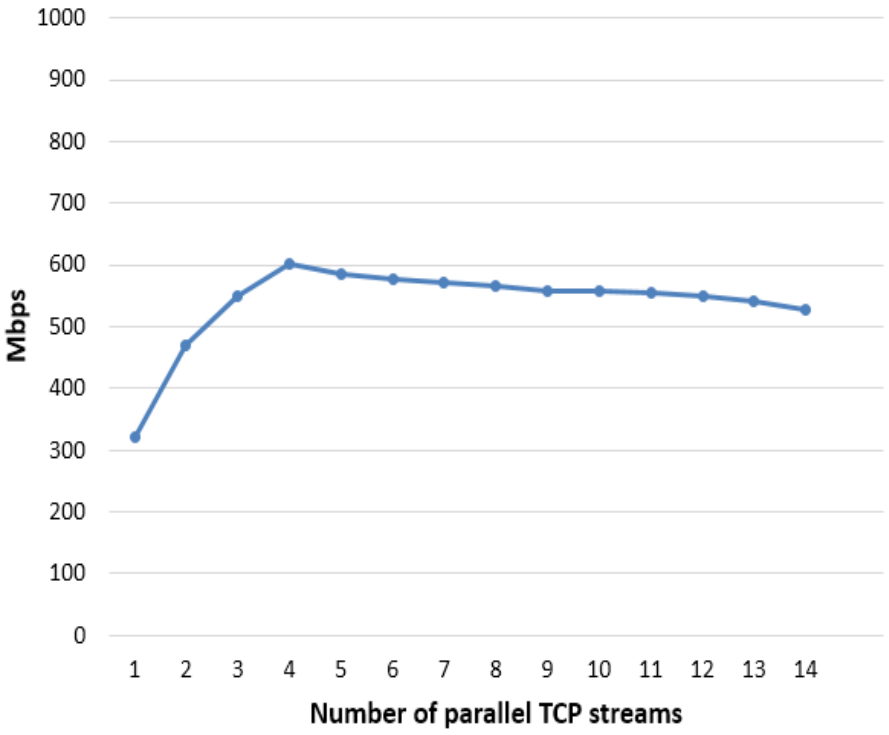

Figure 4.1: Relationship between achieved bandwidth and number of parallel TCP streams in high Bandwidth-Delay Product networks (An experimental result of data transfer from our network testbed)

Figure 4.1 shows an experimental result of data transfer from our network testbed. In this case, when we increase the number of parallel TCP streams, the achieved bandwidth also increases. However, when we increase the number of parallel TCP streams more than 4, the achieved bandwidth stops increasing and begin to decrease slowly.

The reason for this phenomenon is with the increasing of the number of parallel TCP streams, the overall rate of recovery from packet loss will increase until the congestion occurs in the network. After this critical point, the number of TCP streams and the amount of congestion become affecting the packet loss rate. The increasing of packet loss rate indicates that the network is congested, and the TCP sender should reduce its congestion window. If we continuously increase the number of parallel TCP connections, the higher packet loss rate will decrease the impact of multiple TCP streams, the aggregate TCP bandwidth will stop increasing, or begin to decrease. Therefore, calculating the critical point is the key point of the prediction model.

4.2 Proposed Prediction Model

There are several existing researches to predict the maximum network throughput with multiple TCP streams. However, little research has been conducted to find the optimal number of TCP streams that can achieve the maximum throughput. Therefore, we derived the prediction model for an optimal number of TCP streams based on some existing models for the network throughput.

According to Hacker et al Model [103], when an application opens a single stream, the maximum network throughput can be represented as:

$$Th <= \frac{MSS}{R}\frac{c_0}{\sqrt{p}}. \tag{4.1}$$

Th represents the maximum throughput, MSS is the maximum segment size of TCP, R is the round trip time, p is the packet loss rate and c_0 is a constant.

Hacker et al [103] also claim that the aggregated throughput of parallel streams can be calculated with the throughput of a single stream multiplied by the number of streams. In addition, Dinda et al [104] show p would increase as the number of parallel streams increases and the network gets congested. Therefore, Eq. (1) can be rearranged for n streams as:

$$Th_n <= \frac{MSS \times c_0}{R}\left(\frac{n}{\sqrt{p_n}}\right). \tag{4.2}$$

n represents the number of parallel streams, p_n is the packet loss rate when

n streams are used on the network. If we use too many streams, p_n increases dramatically and Th_n decreases.

According to [105], the packet loss rate in a network, p_n, is determined only by the available bandwidth(B)-latency(R) product per TCP connection (i.e., BR/n) and can be represented as:

$$p_n = \left(c_1 \left(\frac{BR}{n} \right)^2 + c_2 \frac{BR}{n} - c_3 \right)^{-1}. \tag{4.3}$$

c_1, c_2 and c_3 are constant and positive numbers.

After placing p_n in Eq. (2), the total achievable throughput Th_n is calculated as follow:

$$Th_n <= \frac{MSS \times c_0}{R} \left(\frac{n}{\sqrt{\left(c_1 \left(\frac{BR}{n} \right)^2 + c_2 \frac{BR}{n} - c_3 \right)^{-1}}} \right). \tag{4.4}$$

Since Eq. (4) is a convex upward function, we can get an optimal n that maximizes the Th_n by solving the following partial differential equation for n:

$$\frac{\partial Th_n}{\partial n} = 0. \tag{4.5}$$

If we assume MSS is a relatively static value, the solution of Eq. (5) is given by the following equation:

$$n = \frac{c_1}{2c_3} BR. \tag{4.6}$$

Since c_1 and c_3 are constants, Eq. (6) can be simplified as follows with a single constant value, a:

$$n = aBR. \tag{4.7}$$

Since n is actually the number of TCP streams, it should be equal to or greater than 1. The equation can be therefore written as:

$$n = max(1, aBR). \tag{4.8}$$

This result seems to be too simplified. However, we got this result by just combining the existing known models for the maximum aggregated bandwidth

and the packet loss rate, as calculated in the above. Also, this result matches intuitive expectations. If we have a larger bandwidth-delay product the number of optimal parallel TCP streams will increase accordingly.

The constant values, from c_1 to c_3, which define the packet loss rate, are determined by the characteristics of the network we use. Therefore, the constant value, a, in Eq. (8) will be also determined by the characteristics of the network. To determine the value, a, we need to measure several combinations of n, B and R. We will verify the prediction model in the next section.

4.3 Verification of Prediction Model

Latency (ms)	Number of TCP streams							
	1	2	3	4	5	6	7	8
0	374.66	374.77	374.66	374.67	374.89	-	-	-
20	376.10	375.02	375.45	375.55	375.44	375.74	375.89	-
40	387.38	376.76	376.30	376.36	376.75	376.40	376.65	376.98
60	390.40	385.32	378.94	377.10	377.47	377.58	377.26	377.48
80	391.03	384.41	381.96	379.30	378.09	378.43	378.14	378.43
100	393.63	387.84	384.26	382.46	381.21	379.72	378.44	378.91

Figure 4.2: Transfer time (sec) of 2GB file with 50 Mbps link

To verify the validity of the prediction model, we have conducted a large amount of data transfer experiments with various network conditions by changing available bandwidth and latency. For the data transfer experimental environment, we prepared two virtual machines with two CPU cores and 2 GB memory on different physical VMware ESX machine equipped with two Intel Xeon E5649 processors and 48 GB memory. The physical machines were connected with a single 1 Gbps network switch. Traffic control tool (tc) of Linux [106] was used to configure the available bandwidth and latency between the two virtual machines.

In order to calculate the constant value, a, of Eq. (4.8) in our virtual environment, we have measured the time needed to transfer a file of 2 GB from one host to another host under different conditions where the available bandwidth is limited to 50 Mbps or 100 Mbps with changing the added latency from 0 to 140 ms.

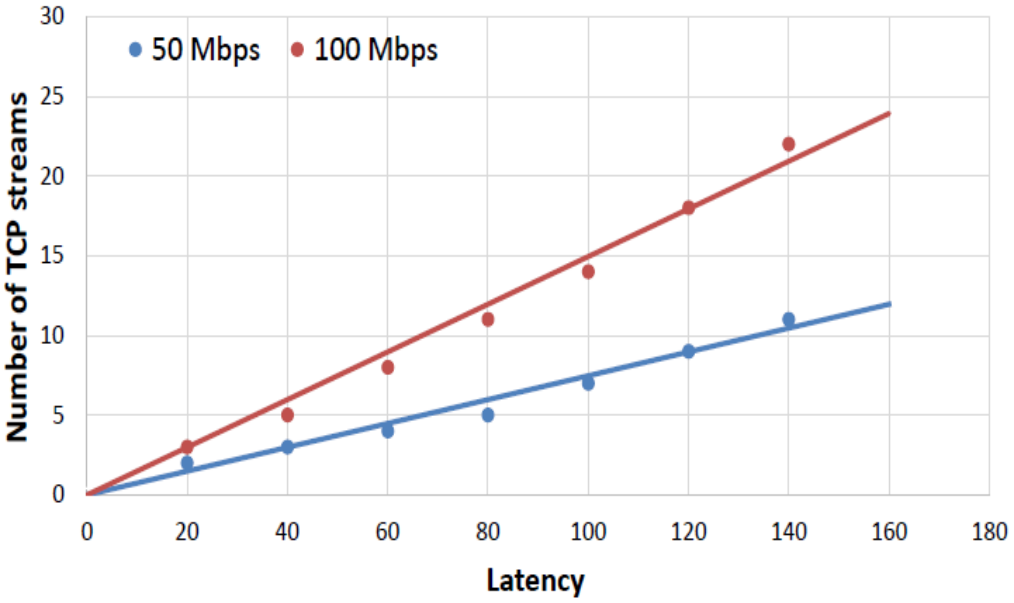

Figure 4.3: Relationship between the optimal number of TCP streams and the latency

Figure 4.2 shows the part of the observed data in the case where the available bandwidth is limited to 50 Mbps. We have repeated this measurement 12 times for each case. The data presented in Figure 4.2 reflects the average of 10 trials excluding the highest and lowest ones. In the Figure, the best results for each different latency are highlighted with red. For example, we can see that using 4 TCP streams achieved the best performance where the latency is configured to 60ms.

Using the measurement results, we can calculate the constant value, a. From the result of the average value calculated with the measurement results, we determined that a is about 0.001495 in our virtual environment. And at section 6.1,

we will use this parameter to conduct experiments in our virtual experimental environment.

Figure 4.3 plots the measured optimal numbers of TCP streams and the predicted lines based on our proposed model. We can see that the measured values are very close to our prediction results. The result also demonstrates the effectiveness of our proposed prediction model. Therefore, by using Eq. (4.8), we can calculate the optimal number of TCP streams if the available bandwidth and latency are given.

5 Use Case of SDN-enabled GridFTP

We explained the implementation of SDN-enabled GridFTP and an optimal TCP assignment prediction model in the previous section. In this chapter, we introduce the target and mechanism of our proposed system.

5.1 Taget of SDN-enabled GridFTP

As we explained in the section 1.2, our proposed system aims to provide a high-speed data transfer service to the computational science research projects. Especially, some data-intensive scientific projects consume significant networking resources which are supported by the specialized network, such as NRENs. However, traditional network routing techniques limit the performance of data transfer. Therefore, to meet the needs of these scientific projects when using specialized network resources, our proposed system, SDN-enabled GridFTP utilizes multiple paths simultaneously and optimization TCP assignment prediction model to achieve the high-speed transfer of large-scale data.

In the current implementation of SDN-enabled GridFTP, we only consider one user utilizing high-speed data transfer service. It is because SDN-enabled GridFTP calculates the optimal TCP streams number for each network path to fully utilize network resource, and it will cause unfairness to the other network users. We will discuss this problem in the section 7.4

5.2 Mechanism of SDN-enabled GridFTP

In order to utilize our proposed system, users need to go through a two-step process. The first step is acquiring available multiple paths using the AssignMultipath; the second step is the deployment of actual flow entries using MakePath. Since the first step for acquiring multiple paths is a preprocessing step before the actual communication starts, we created a lightweight client program called assign_mpath just for calling the AssignMultipath interface. Also, we implemented an XIO driver to access the MakePath interface for the second step and implemented the XIO driver to be called from GridFTP.

The actual execution procedure of program is as follows:

```
> ./assign_mpath <controller_address> <port> <src_ip>
<src_mac> <dst_ip> <path_num>

> MPATH_ASSIGNMENT_ID=<id> globus-url-copy -p <path_num>
 <Source_URL>  gsiftp://<Destination_URL>
```

The *assign_mpath* program requires the IP address of the multipath OpenFlow controller (`controller_address`), the port number of the controller (`port`), the source IP address (`src_ip`), the source MAC address (`src_mac`), the destination IP address (`dst_ip`) and the number of paths needed to be assigned (`path_num`). *assign_mpath* outputs the number of assigned paths and an assignment ID. This assignment ID is a unique ID identifying the assigned paths and will be used later for calling the *MakePath* interface.

The *globus-url-copy* is a GridFTP client provided by the Globus Toolkit. It is a scriptable command line tool that can do multi-protocol data movement. The option *p* with `path_num`) specifies the number of parallel data connections that should be used. The `Source_URL` specifies the original URL of the file(s) to be copied. If this is a directory, all files within that directory will be copied. The option `gsiftp://` exactly specifies the GridFTP protocol as the data transfer protocol. The `Destination_URL` specifies the URL where you want to copy the files. In addition, we use an environment variable, `MPATH_ASSIGNMENT_ID` in order to give the assignment ID to our developed XIO driver. The *globus-url-copy* is therefore launched with the variable, `MPATH_ASSIGNMENT_ID`.

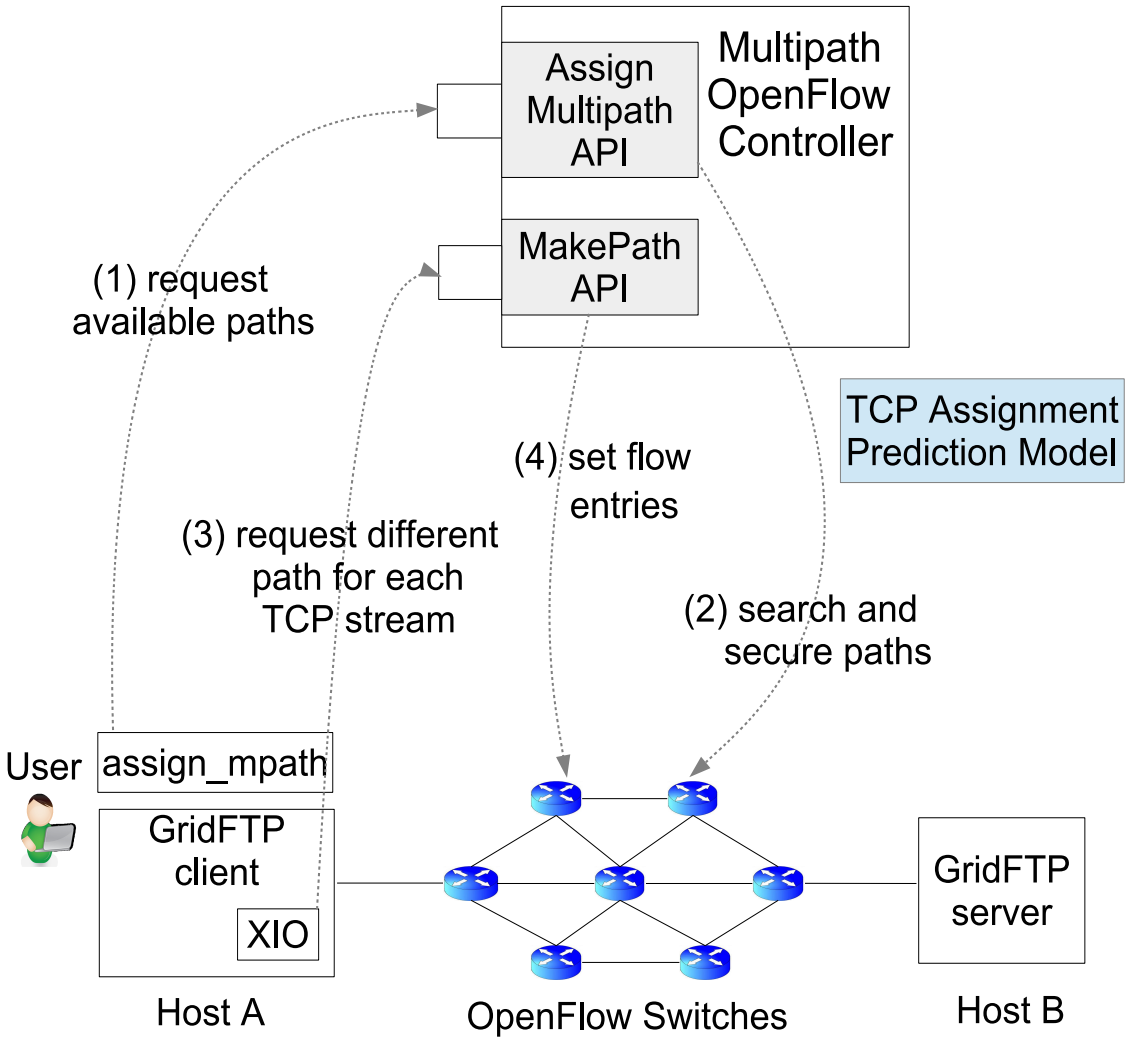

Figure 5.1: Overview of our proposed multipath controller and GridFTP

Figure 5.1 illustrates the proposed system. The proposed system performs the following steps:

- First, in (1), a user launches the *assign_mpath* program. It accesses the *AssignMultipath* interface implemented on our multipath OpenFlow controller, and requests available paths and optimal parallel TCP streams from Host A to Host B.

- Next, in (2), the *AssignMultipath* calculates the available paths with the breadth-first search algorithm from Host A to Host B based on the topology of the OpenFlow network. The *AssignMultipath* secures the specified number of available paths and returns the number of paths with an assignment ID to the user. The assignment ID will be used to refer to the assigned paths for later use. The *AssignMultipath* also calculates the optimal number of TCP streams for each secured path according to optimal TCP stream assignment prediction model. The total number of each path's optimal TCP streams will be also returned to the user for specifying the option p of *globus-url-copy*.

- Then, the user starts the GridFTP client with the assignment ID and specifies the number of parallel TCP streams obtained in the previous step.

- In (3), our implemented XIO driver loads the assignment ID. Then, the XIO driver accesses the *MakePath* interface on the OpenFlow controller and requests to create an individual path each time the GridFTP client opens a TCP stream via the XIO driver.

- In (4), the *MakePath* finds the set of paths assigned by the *AssignMultipath* using the assignment ID and acquires a path from the path set, and then installs flow entries into the OpenFlow switches for the requested TCP stream. The *MakePath* creates flow entries using the source and destination IP addresses and the source TCP port number as match conditions and installs the flow entries to each OpenFlow switch for the path.

- Finally, the requested TCP stream starts the communication according to the assigned route. Steps (3) and (4) are applied repeatedly for the subsequent TCP streams.

6 Evaluation and Results

To verify the effectiveness of the proposed system, we performed evaluations comparing the performance of the GridFTP with as well as without our proposed method. For retrieving the best possible performance, we conducted the evaluations in a virtual environment first. We then also performed some evaluations on a real global-scale environment to evaluate the practicality of our proposal.

6.1 Experiments using a Virtual Environment

In our virtual environment, we conducted our experiments over a simple topology to confirm if our proposed system can achieve the expected results. Furthermore, we designed another more realistic topology, which has some overlapped links among different paths, to verify the effectiveness of our system.

6.1.1 Virtual Experimental Environment

Figure 6.1 shows the overview of the experimental virtual environment. We installed the GridFTP on two machines: Host A and Host B, and used these two machines as a client and a server respectively. In this experiment, due to lack of hardware OpenFlow switches resources, we have prepared multiple hosts called, Switch Hosts, installing software-based implementation of OpenFlow switch, Open vSwitch [107]. We deployed these Switch Hosts between Host A and B, and constructed an OpenFlow network which has multiple paths between Host A and B.

Our experimental virtual environment is constructed on six virtual machines deployed on each of six physical VMware ESX machines equipped with two Intel Xeon E5649 processors and 48GB memory. We assigned two virtual cores and 2GB memory for each virtual machine, and setup CentOS 6.5 on each of them.

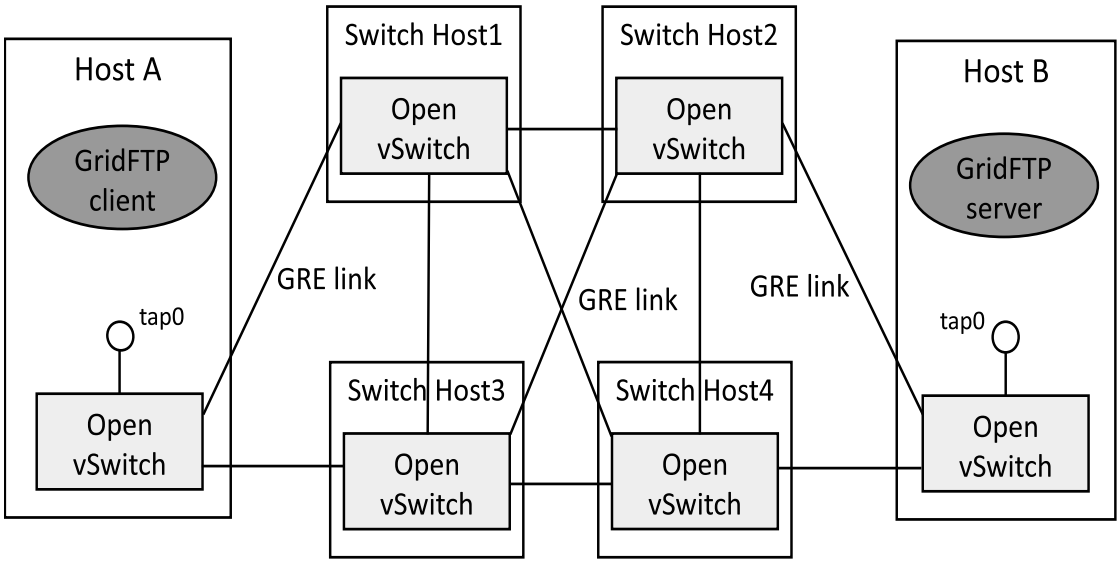

Figure 6.1: Overview of the virtual experimental environment

These virtual machines share a single 1Gbps network switch physically, and the actual available bandwidth is about 941Mbps measured by iperf between two hosts.

In addition, in order to make the communication of GridFTP pass through the OpenFlow network, we also installed Open vSwitch on Host A and B, and added a virtual network device, tap on them. The Open vSwitches on Host A, Host B and other Open vSwitches are connected by GRE [108] links, which is an IP-based point-to-point tunneling protocol. By changing the combination of the GRE links, we can easily construct various topologies for the experiments and configure each path with different bandwidth and latency.

In this virtual environment, we conducted the experiments on three network topologies, topology A, B and C. Topology A (Figure 6.2) is a simple topology which has four independent paths, and the bandwidth and latency of each path are configured to 100 Mbps and 0 ms respectively. Topology B (Figure 6.5) has same configured bandwidth and latency with topology A. However, it has more realistic topology. Topology C (Figure 6.7) assume a more realistic situation. It

has some overlapping links and different bandwidth and latency are configured. In the figures, SW1 to SW6 denote Open vSwitches.

6.1.2 Results of Experiments

In the experiments, the data transfer time was measured. In our proposed system, it is necessary to run the *assign_mpath* command in advance to find the routes. But, we did not measure the time taken for *assign_mpath*, since these experimental topologies are very small and the required time for executing *assign_mpath* is very short. We leave such evaluation on the scalability of *assign_mpath* with larger and more complex topologies as a future issue.

In addition, we measured the used bandwidth by periodically monitoring packet counters on the OpenFlow switches. Each packet counter on OpenFlow switches records the number of transferred packets and transferred bytes for a flow-basis. Thus, this information is useful to measure the bandwidth of each TCP stream of the GridFTP separately. We measured the counter in the Open vSwtich on Host B, which is the destination of the data transfer. As we calculated the parameter of Eq. (4.8) in the section 4.3, we will use the result to calculate the optimal number of TCP streams in our virtual environment.

Results of Topology A

In topology A, we conducted two experiments in the case of two and four parallel TCP streams by transferring a file of 1Gbyte. In this topology, there are four available paths, 1) $SW1$-$SW2$-$SW6$, 2) $SW1$-$SW3$-$SW6$, 3) $SW1$-$SW4$-$SW6$ and 4) $SW1$-$SW5$-$SW6$. In this paper, $SW1$-$SW2$-$SW6$ represents a path which walks through switches $SW1$, $SW2$, $SW6$ in this order. The optimal number of TCP streams is calculated as 1 for each path in this topology because the added latency for each path is configured to 0 ms.

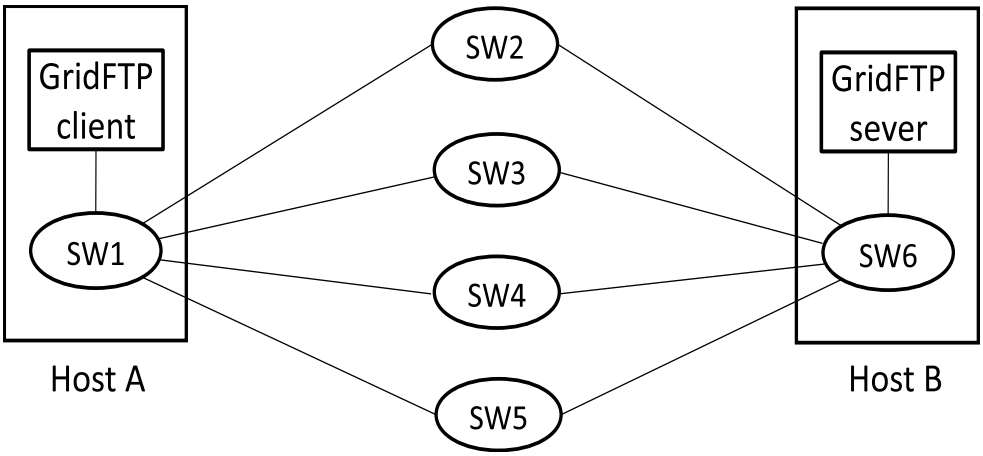

Figure 6.2: Topology A on a local testbed (the bandwidth and the latency are configured to 100 Mbps and 0ms on each path)

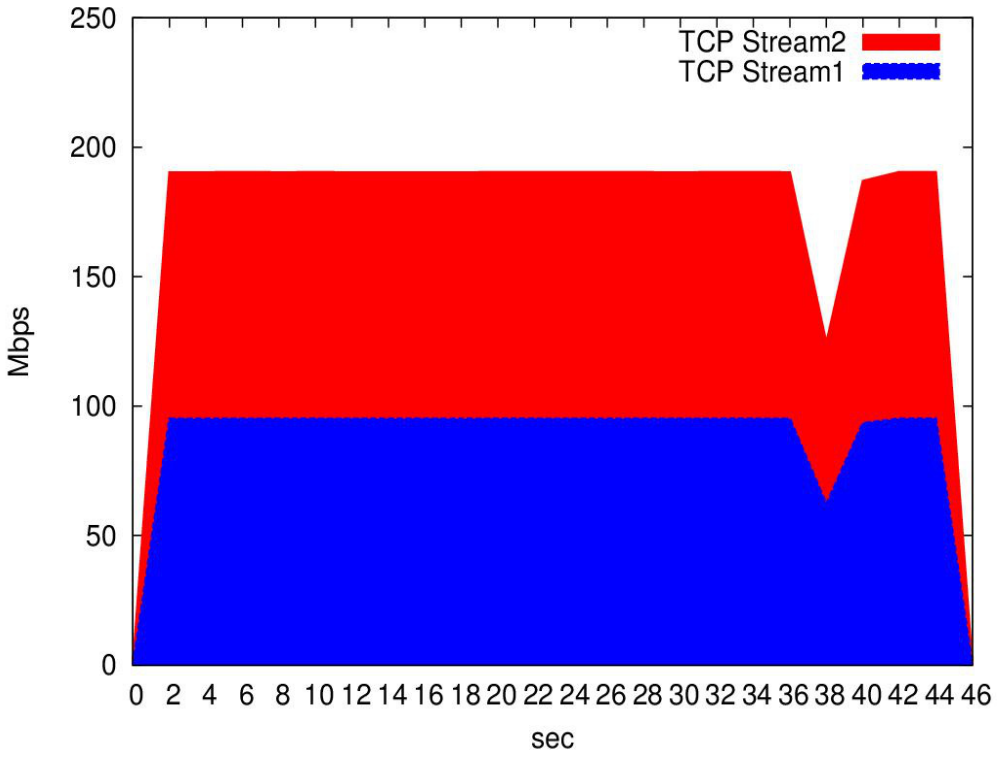

Figure 6.3: Used bandwidth of each TCP stream in topology A with two parallel TCP streams

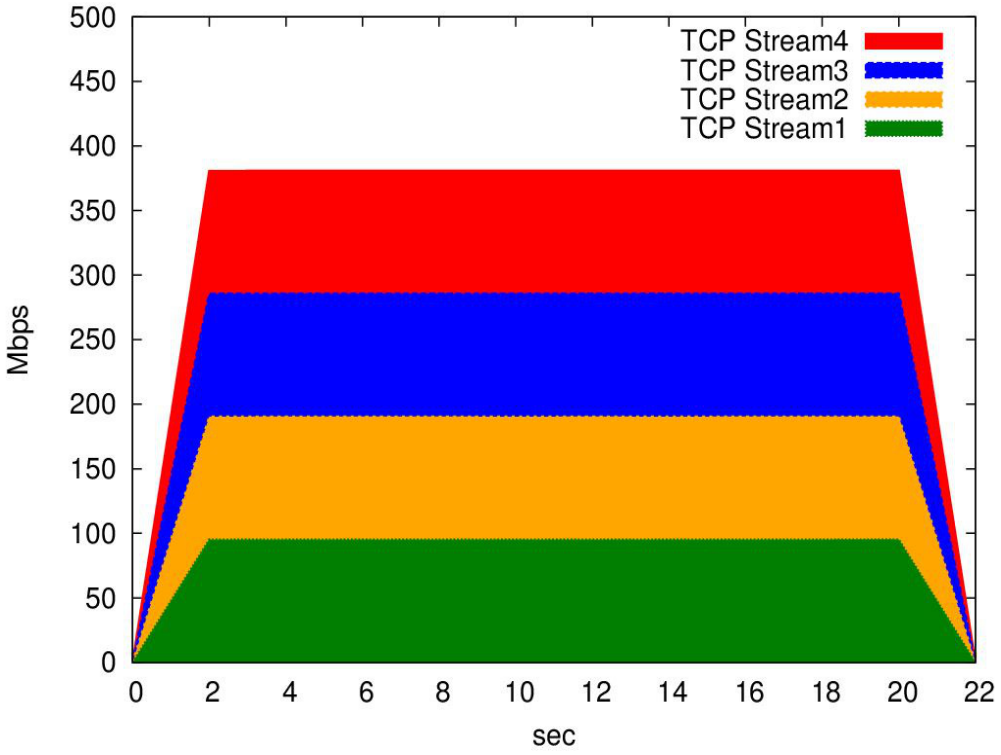

Figure 6.4: Used bandwidth of each TCP stream in topology A with four parallel TCP streams

Number of streams	Proposed multipath system	Conventional single path method
2	47.486s	92.108s
4	23.404s	90.533s

Table 6.1: Comparison of transfer time in topology A

Table 6.1 shows the experimental results of Topology A. The result shows that our proposed system successfully distributes multiple TCP streams of GridFTP into different paths and improved data transfer speed, while the conventional single path method just uses a single path for all four TCP streams. In the case using two parallel TCP streams, the proposed system shortened the transfer time by about half. In addition, in the case using four parallel TCP streams, the proposed system shortened the transfer time about one-quarter. Figure 6.3 and

Figure 6.4 show the used bandwidth summarized as stacked area graphs which are calculated from the average transferred bytes per second. We can see that the performance of each TCP stream is also very stable.

Results of Topology B

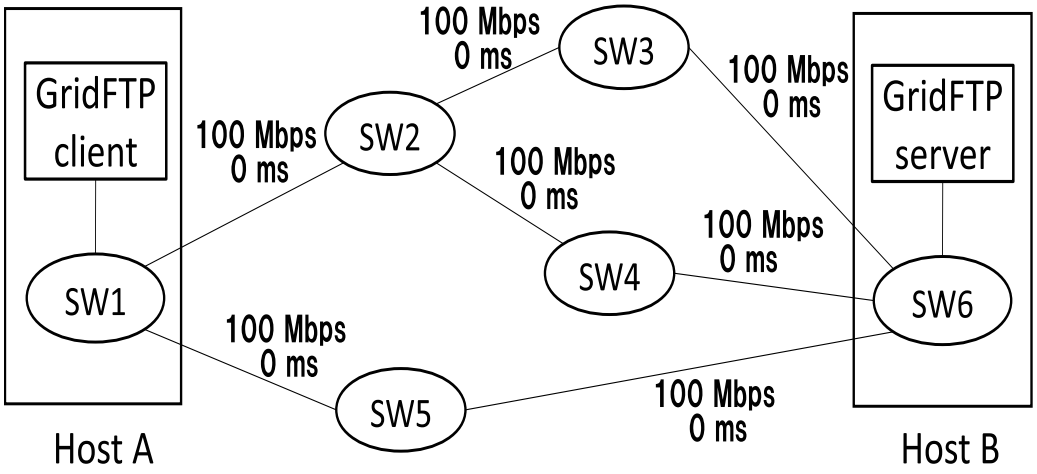

Figure 6.5: Topology B on a local testbed (the bandwidth and the latency are configured to 100 Mbps and 0ms on each path)

In topology B, there are three available paths, 1) sw1-sw5-sw6, 2) sw1-sw2-sw3-sw6 and 3) sw1-sw2-sw4-sw6. But, path 2) and 3) share a link, sw1-sw2. We conducted one experiment using three parallel TCP streams in this topology.

Number of streams	Proposed multipath system	Conventional single path method
3	46.603s	1m30.480s

Table 6.2: Results of Topology B

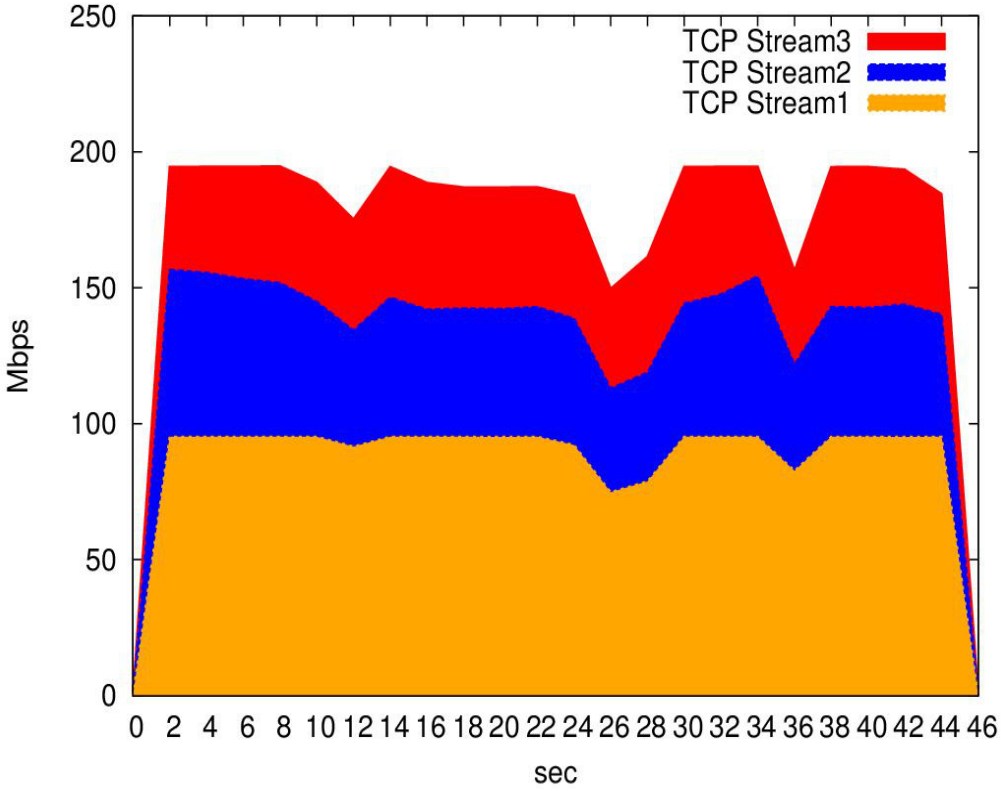

Figure 6.6: Used bandwidth of each TCP stream in Topology B with three parallel TCP streams

Table 6.1.2 shows the experimental results of Topology B. The results show that our proposed system shortened the transfer time by about half. Since we used one independent path and two paths sharing a link, the improvement of the data transfer has only doubled. Figure 6.6 shows the used bandwidth of each TCP stream. TCP Stream1, 2 and 3 used the path 1), 2) and 3) respectively. Since the path for TCP Stream1 was independent of the other two paths, the bandwidth for the path is about 95Mbps and the performance was very stable. On the other hand, the paths for TCP Stream2 and 3 shared a link, sw1-sw2, and the bandwidth was about half of the TCP Stream1. In addition, the performance of the data transfer was also not stable compared to that of TCP Stream1. From these experiments, we found that our proposed system works correctly as expected.

Results of Topology C

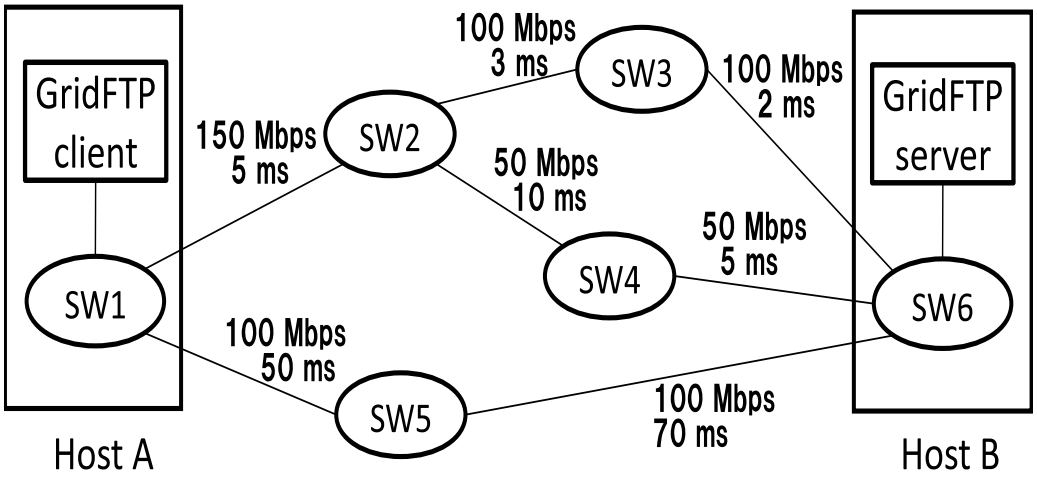

Figure 6.7: Topology C on a local testbed (different bandwidth and latency are configured for each path as shown in the Figure)

In topology C, we compared the proposed optimal assignment and a simple round robin assignment by transferring a file of 10 Gbyte to evaluate the efficiency of the proposed optimal assignment method. There are three available paths, 1) $SW1$-$SW2$-$SW3$-$SW6$, 2) $SW1$-$SW2$-$SW4$-$SW6$ and 3) $SW1$-$SW5$-$SW6$ in the topology. With the proposed optimal assignment, the assignment of TCP streams for each path is decided based on the calculated optimal number of TCP streams of each path. On the other hand, with the simple round robin assignment, the assignment of TCP streams for each path is equally distributed. In the optimal assignment method, the optimal assignment of TCP streams is calculated as that 2 streams for each first and second path and 18 streams for the third path. Therefore, using 22 TCP streams in total is the optimal number of streams in this case.

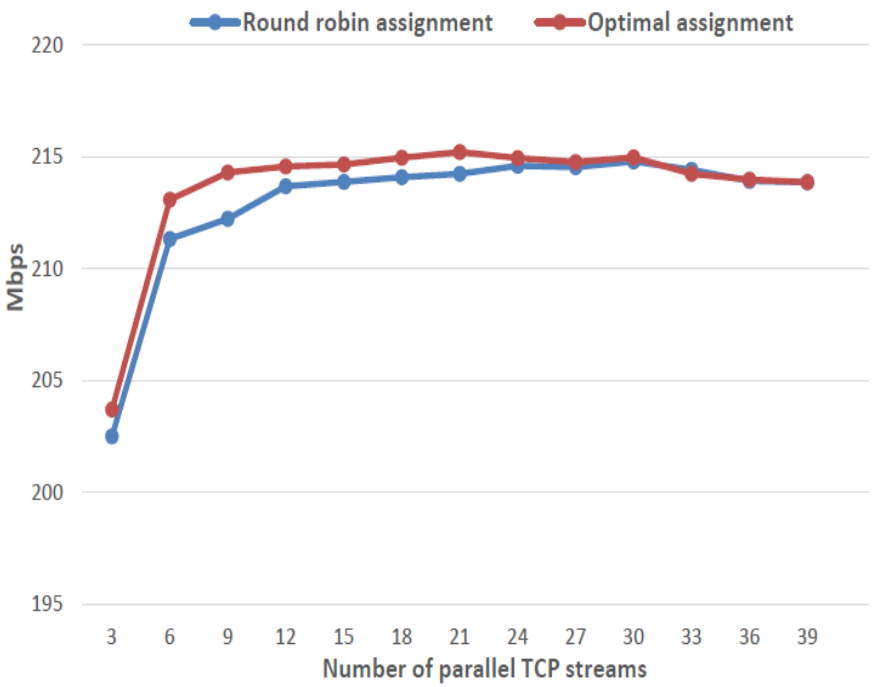

Figure 6.8: Comparison of the average data transfer speed between the optimal assignment and the round robin assignment in topology C

To compare the performance of our optimal method and the simple round robin method, we actually measured the transfer time with increasing the number of TCP streams from 3 to 39 by 3. Since the optimal assignment of streams is 2, 2 and 18 streams for each of three paths, we assigned the TCP streams to the three paths with a ratio of 1:1:9 in the proposed method. On the other hand, we assigned the TCP streams in a round robin manner for the simple round robin method. We repeated the same experiment 10 times and calculated the average for the results.

Figure 6.8 shows the average data transfer speed of the optimal assignment and the round robin assignment with increasing the number of parallel TCP streams. As shown in Figure 6.8, the proposed optimal assignment method achieves an overall better throughput than the round robin assignment. This also means that the performance of the optimal assignment method is converged to the peak performance more quickly. Also, the peak of the performance is located in the

position where the number of TCP streams is around the predicted optimal number, 22. The results demonstrated the effectiveness of our proposed system with the proposed optimal TCP stream assignment.

6.2 Experiments using a Real Global-scale Environment

To evaluate the practicality of our method, we have also conducted some experiments in a real global-scale environment.

6.2.1 PRAGMA-ENT

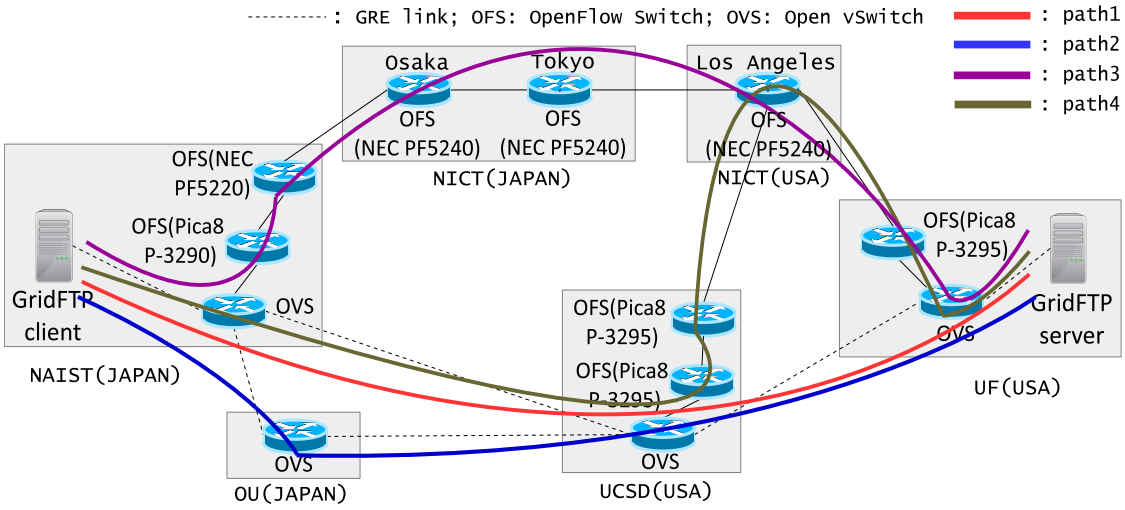

Figure 6.9: Overview of the real global-scale experimental environment

For the evaluation, we used the resources provided by PRAGMA Experimental Network Testbed (PRAGMA-ENT) [109] [110]. PRAGMA-ENT provides a large scale OpenFlow network composed of computing resources and international academic networks. The part of resources is also connected through GRE over the public Internet as alternative paths. We used a part of the resources provided by PRAGMA-ENT.

Figure 6.9 shows the overview of the experimental environment. We installed GridFTP on a virtual machine as a client and deployed an Open vSwitch and two hardware OpenFlow Switches (Pica8 P-3290 and NEC PF5220) at Nara Institute of Science and Technology, Japan (NAIST). Also, we installed GridFTP on a virtual machine as a server and deployed an Open vSwitch and a hardware OpenFlow Switch (Pica8 P-3290) at the University of Florida, USA (UF).

There are three sites between NAIST and UF: 1) National Institute of Information and Communications Technology, Japan (NICT) that deployed three hardware OpenFlow Switches (NEC PF5240): one is in Osaka Data Center, one is in Tokyo Data Center and another one is in Los Angeles Data Center, 2) Osaka University, Japan (OU) that deployed an Open vSwitch, 3) University of California, San Diego, USA (UCSD) that deployed two hardware OpenFlow Switches (Pica8 P-3290) and an Open vSwitch.

The experimental environment uses different international and academic networks and GRE : 1) The GRE connection between NAIST and UCSD is established over the TransPAC3 network; 2) The GRE connections between OU and NAIST, OU and UCSD are established over Science Information NETwork, Japan (SINET5); 3) NAIST, Osaka, Tokyo and Los Angeles Data Center of NICT are connected with RISE service over JGN-X; 4) The links between UF and Los Angeles, UCSD and Los Angeles, UCSD and UF are connected via Internet2 and California Research and Education Network (CalREN). All experiments on the global environments are conducted during weekends to reduce the impact from the background traffic because there is usually a larger traffic during working days and a smaller traffic during weekends on these national research and education networks.

6.2.2 Determine the Parameter of Prediction Model on PRAGMA-ENT

Path number	1	2	3	4
Latency (ms)	191	183	211	191
Available average bandwidth (Mbps)	728.4	638	916	780.5
S.D. of available bandwidth	0.78	3.82	4.97	20.43
Observed optimal number of streams	8	10	10	28
Predicted optimal number of streams	9	8	13	10

Table 6.3: Experiment result to determine value a in the global-scale experimental environment

In order to determine the value, a, of Eq. (8) in our global-scale environment, we have also measured the available bandwidth and latency on several paths and figured out the optimal number of TCP streams for those paths. Figure 6.9 shows four shorter paths (path1 to 4) that our system found. Since longer paths than these four paths have too many overlapped links with the other paths and are not useful to evaluate, we use the four paths illustrated in Fig. 6.9.

The Table 6.3 indicates the measurement results of the four paths. As the fourth path has a bigger standard deviation and the performance is not stable on the path, we use the other three paths to calculate the value, a. From the measurement results, we have determined the value, a as about 0.000065 in our international environment. The Table 6.3 also shows the predicted optimal number of TCP streams. We can see that the numbers are slightly different from the observed optimal numbers but still similar.

6.2.3 Results of Experiments

In the experiments, we compared our proposed system with two other methods, 1) the single path assignment method which is a conventional routing method just using a single path for multiple TCP streams, 2) the round robin assignment method which uses available multiple paths in a round robin manner. Our method uses available multiple paths based on the rate from the predicted optimal numbers of TCP streams for each path. For the evaluation, we transferred a file of 6GB from NAIST to UF, and measured the transfer time and also measured the used bandwidth during the transfer. The measurement method is the same as the method used in our virtual environment experiments.

Path number	1	2	3	4
Latency (ms)	191	183	211	191
Expected available bandwidth (Mbps)	660	200	740	170
Optimal number of streams	8	2	10	2

Table 6.4: Optimal assignment in the global-scale experimental environment

Since the used four paths have shared links each other, the available bandwidth would be reduced when these four paths are used simultaneously. We measured a standalone performance of each link and expected the available bandwidth of each path as shown in Table 6.4. Based on the expected bandwidth, we have calculated the optimal number of TCP streams for each path as 8, 2, 10 and 2 respectively. Therefore, we assigned TCP streams to the four paths with a ratio of 4:1:5:1 in our optimal assignment method.

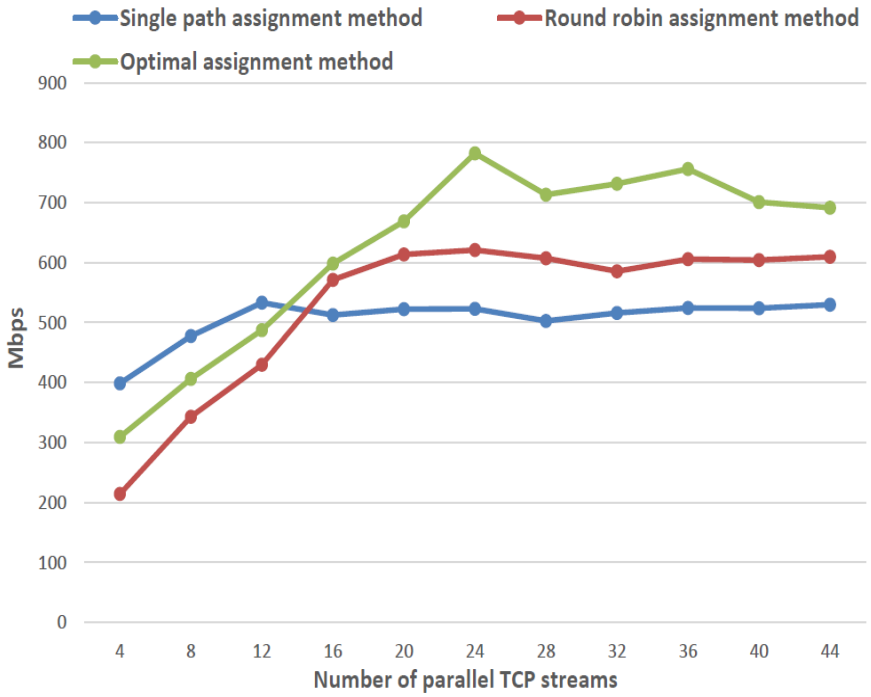

Figure 6.10: Comparison of the average data transfer speed between the single path assignment, the round robin assignment and the optimal assignment

Figure 6.10 shows the average speed of the data transfer while increasing the number of parallel TCP streams. From the results, in the case of using 4, 8 and 12 parallel TCP streams, the average speeds of the single path assignment method are better than the optimal assignment and the round robin assignment method. This is because our proposed system used path1, 2, 3 and 4 simultaneously, and only a few TCP streams were assigned for each path in the case of using smaller streams. Therefore, those TCP streams could not overcome the performance degradation of TCP's slow start mechanism.

On the other hand, in the case of using the single path assignment method, all streams were assigned to the shortest path, path1, and achieved better performance than our method. However, when we used more than 16 streams, the optimal assignment and the round robin assignment method achieved better performance than the single path assignment. Especially, the maximum performance

of our optimal assignment method reaches approximately 30% better than the round robin assignment method and approximately 60% better than the single path assignment method.

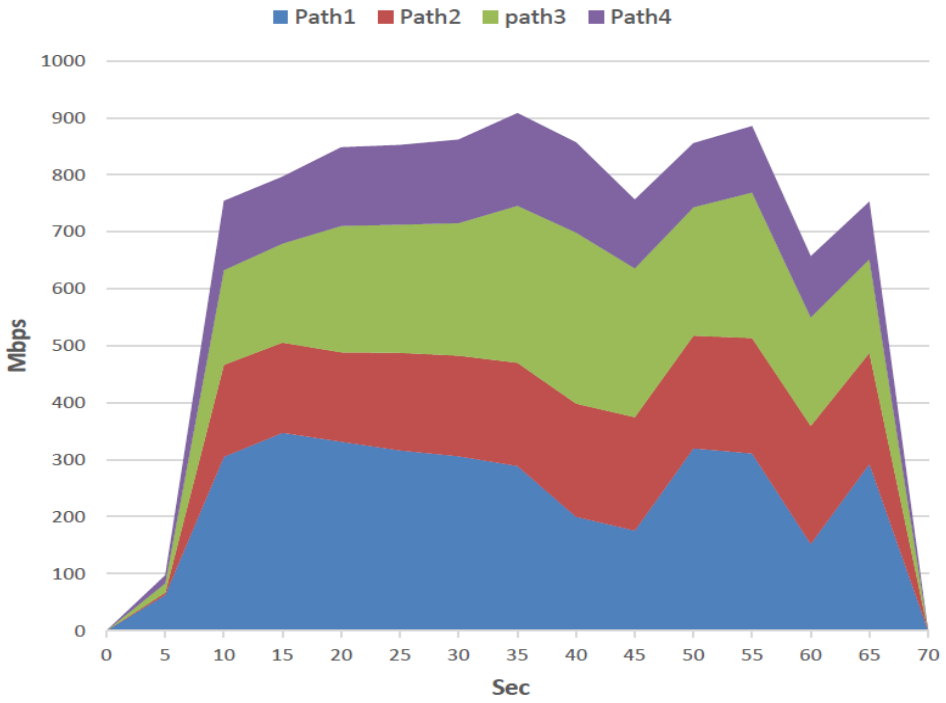

Figure 6.11: Used bandwidth for the round robin assignment method in case of using 24 parallel TCP streams (6 parallel TCP streams are assigned for each path)

Figure 6.11 shows the used bandwidth for the round robin assignment method with 24 parallel TCP streams. The performance for path4 is slightly worse than the other paths because path4 is the longest and unstable path. The results show that the bandwidth keeps at around 880 Mbps and also achieved 900 Mbps as its best performance.

Figure 6.12 shows the used bandwidth for the optimal assignment method with 22 parallel TCP streams. The performance of path2 and 3 is worse than that of the other paths. But, the overall performance of the aggregated bandwidth is larger than the round robin assignment method. The results show that the

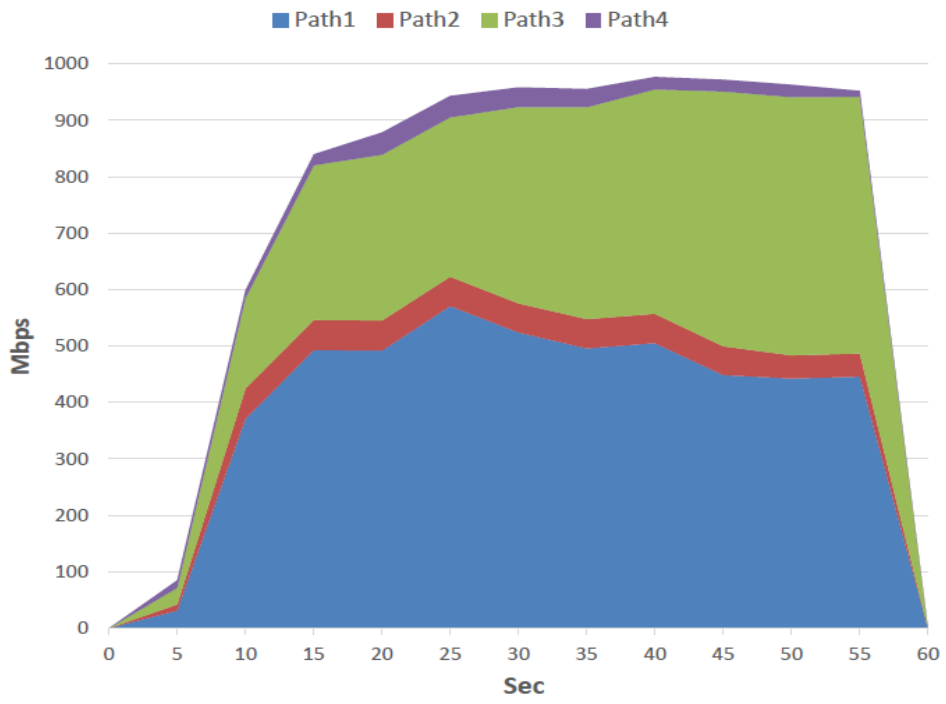

Figure 6.12: Used bandwidth for the optimal assignment method in case of using 22 parallel TCP streams (TCP streams are assigned for each path with a ratio of 4:1:5:1 in order)

traffic is stable and the bandwidth keeps at around 960 Mbps.In this experiment, our virtual machine host was equipped only with a 1 Gbps NIC. So, this result indicates that our proposed method achieved the performance that is close to the physical limitation of the hardware.

In addition, the network congestion happened in both the round robin method and the optimal assignment method. As Figure 6.11 shows, two congestion points appear at 35th and 55th second. The performance of each path degrades after the congestion. The reason can be considered that the congestion happened at UCSD site in figure 6.9. Since the round robin method does not consider the network conditions and assigns the same number of TCP streams on each path. We use 6 parallel TCP streams in path 1, 2 and 4, so 18 TCP streams go through the ovs of the UCSD site. The ovs cannot process all packets when total windows

size of 18 TCP streams increase too big and cause packet loss.

In figure 6.12, there is also a congestion point at 25th second. The performance of path 1 and 2 begin to degrade. We consider the congestion also occurs in the ovs of the UCSD site. One possible reason is that path4 is the longest and unstable path as we mentioned above. However, the performance of the path 3 and 4 continues to increase and the total performance keeps increasing steadily. The main reason is we utilized prediction model to assign an optimal set of TCP streams to the different path. So our proposed system and prediction model can be also considered efficient in our global scale network.

7 Discussion and Future Work

This chapter discusses possible issues when actually utilize our proposed system. For future works, we also provide some feasible methods to deal with these possible issues.

7.1 Multipath Selection Algorithm

In our current implementation, we utilized a simple breadth-first searching algorithm since our proposed prediction model can optimize TCP streams according to the network condition of each path. When a user request the specified number of paths, our algorithm just assign the number of paths in the order of hops.

However, each network path has it own maximum attainable throughput. When a user wants to send data as soon as possible with specified number of paths, current algorithm probably could be a problem. Hence, in this case, it is necessary to consider network condition of each path to assign a better multipath set. Furthermore, we also need to consider the overlapping part among paths. We explain our future multipath selection algorithm through two network topologies.

Figure 7.1 shows a network topology. Assume Host A wants to transfer data to Host B. The controller will calculate all available paths according to number of hops. The calculation result should be in the order of path3 (3 hops), path2 (4 hops) and path 1 (5 hops).

Path number	1	2	3
Latency (ms)	60	110	160
Available bandwidth (Mbps)	100	50	50

Table 7.1: Available bandwidth and total latency of each path on topology 1

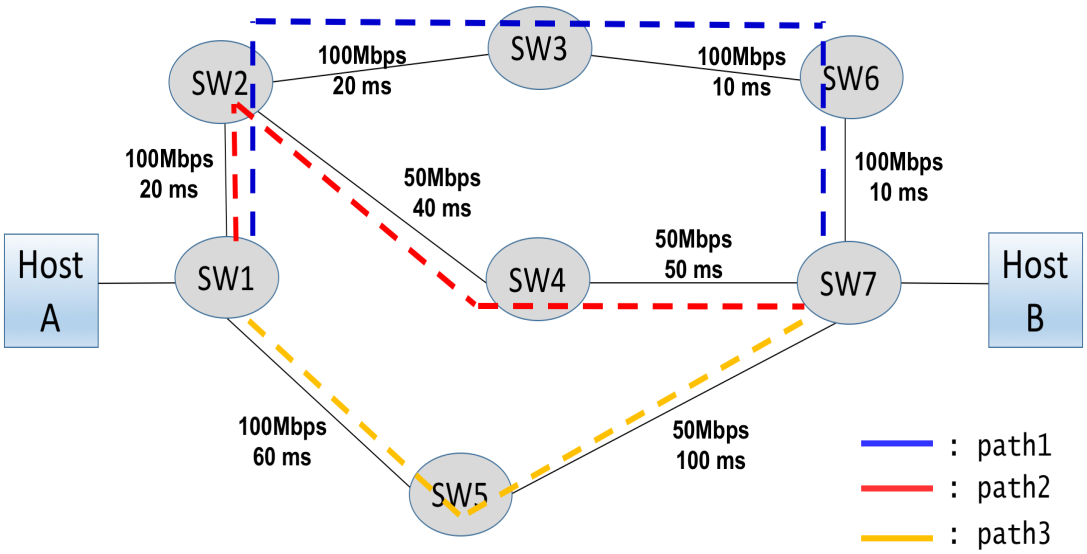

Figure 7.1: Topology 1 (Illustration of multipath selection algorithm)

Table 7.1 shows bandwidth and latency of each path on topology 1. The three paths have different bandwitch and latency. There is no doubt that the path 1 is the best choice. Because it has the higher bandwidth and lower latency in the three paths. In regard to path2 and 3, path 2 and 3 have the same available bandwidth, but path2 has lower latency than path3. Therefore, path selection should be in order of path 1, 2 and 3 when considering the priority.

In addtion, overlapping part is also an important factor should be considered. We can easily choose path 1 when only one path was requested for data tranfer. According the priority, the path 1 and 2 will be chosen when two paths were requested. However, since path 1 and 2 have a overlapping part, if we use path 1 and path 2 simultaneously, the TCP streams assigned for the two paths will compete network resourses between SW1 and SW2. This will cause performance decreasing of data transfer on the contrary. In our future algorithm, a network path which has no overlapping with other paths will be given a higher priority. Therefore, when we choose two paths for GridFTP, path 1 and 3 should be the answer and also the best multipath combination for GridFTP. Because if we use more than two paths in this network topology, the performace may be decrease

due to traffic collision. In this case, even though users specify three or more paths, we only return them the best multipath set and total number of optimal TCP streams.

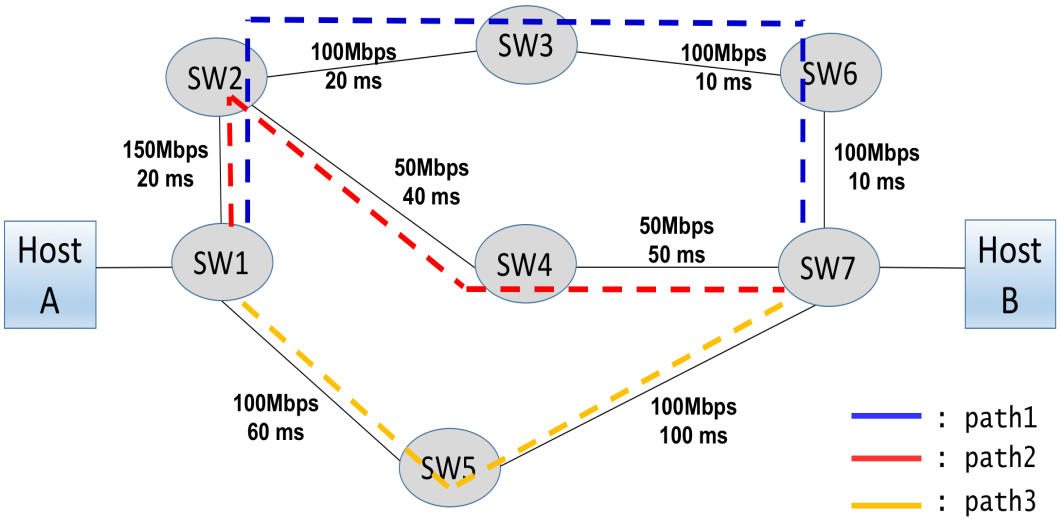

Figure 7.2: Topology 2 (The bandwidth between SW1 and SW2 is different from Topology 1)

Path number	1	2	3
Latency (ms)	60	110	160
Available bandwidth (Mbps)	100	50	100

Table 7.2: Available bandwidth and total latency of each path on topology 2

Figure 7.2 shows topology 2 which has same topology as topology 1. Topology 2 only has some different bandwidth in two parts of network path (Between SW1 and SW2, SW5 and SW7) compared to Topology 1. The bandwidth and latency of each path on topology 2 shown as table 7.1. The path 1 is still the best choice from the three paths. In regard to path2 and path3, path2 has lower latency than path3, but path3 has higher available bandwidth than path2. Since we will use multiple TCP streams to transfer data, we can achieve better performance when

using path3. Therefore, path selection should be in order of path 1, 3 and 2 when considering the priority.

In this topology, when two paths were requested, path 1 and 3 can be chosen. When three paths were requested, though path 1 and 2 share a common part of network path, the sharing part (between SW1 and SW2) has 150 Mbps bandwidth. This bandwidth has enough capacity to make data transfer even using path 1 and 3 simutaneously. Therefore, the best multipath set for this topology is using all three paths.

7.2 Scalability and Reliability of our Multipath OpenFlow Controller

We evaluated our proposed system on a virtual and a real global-scale network environment. The evaluation results showed that our system performed very well. However, the scale of both network environments is not big enough. The practicality and performance of our multipath OpenFlow controller still need to be tested.

Scalability or performance of an OpenFlow controller is usually tested by characteristics of throughput and latency. A scalable controller can keep maximum throughput (number of outstanding packets) and minimum response time even if the number of switches is increased. Regarding the reliability of an OpenFlow controller, when under an average workload, a reliable controller can operate the entire network persistently without accidental connections failure or dropping OpenFlow messages from the switches. Accordingly, we will test our controller by using larger network testbed in the future. As an option, we can also use network simulation tools such as Mininet [111].

The architecture of OpenFlow network provides the OpenFlow controller a global view of the network and makes it easy to achieve an optimal configuration. However, the part of the controller could become a bottleneck in large scale deployment due to such as added latency in newly established flows. Regarding this issue, multiple OpenFlow controller cooperation is a good option. We can deploy multiple controllers in different areas or countries. At the same time, we can make the controllers communicate with each other to control the entire

network.

In addition, as mentioned in section 2.3.2, there are a lot of frameworks to develop OpenFlow controller. A set of performance comparison experiments were conducted to compare ten types of OpenFlow controller which were developed by different frameworks [112]. The results show that the performance of these controllers was totally different. Therefore, we also need to consider this important factor when developing our OpenFlow controller in the future.

7.3 Path Failure

Network path failure happens when a network device (such as switch, router) or a link connection between network devices fail due to planned maintenance or unplanned accidents such as power outages, fiber cuts, and hardware/software malfunction [113, 114]. These path failure issues occur often, everywhere and approximately 80% are unplanned [115, 116]. These routing problems can affect the performance and reliability, especially in common commercial network.

Ideally, the routing system detects unplanned link failures and reconfigures routing tables to avoid using failed links. Some researches showed that interdomain routers may take as long as tens of minutes to reconstruct new paths after a path failure [117].

Assume a path failure occurs during the data transfer of our system, the TCP streams of failure path will timeout and abort. Since our system utilizes multipath to conduct data transfer, even when some TCP streams of GridFTP fail, the data transfer continues.

In addition, we have another option for this issue. Since our system is based on SDN, we can implement a monitoring function of dynamic path failure detection on OpenFlow controller. According to the path routing of the controller, it is also possible to provide other alternative paths for the TCP streams when a failure path is detected.

7.4 Fairness

Fairness is an important factor that should be considered in computer networks. It is normally defined as equal sharing of the network resources. In the current implementation of SDN-enabled GridFTP, we only consider one user utilizing high-speed data transfer service. Our proposed system optimizes multiple parallel TCP streams to fully utilize network resources. However, this approach actually defeats the congestion control [29] mechanisms of TCP, leading to unfairness and potential network congestion.

According to a serious simulation experiment which was performed to test the fairness between single TCP stream and parallel TCP streams on a network path [118], when utilization of available bandwidth is more than 90%, the parallel multiple TCP streams could increase throughput by stealing the bandwidth from competing with single TCP stream. This means when our system fully utilizes a network path, the other single TCP streams will be affected.

Improving the performance of TCP is easy, but improving the efficiency while maintaining fairness on a sharing network is still difficult. There is very few research focused on maintaining fairness by using multiple TCP streams [42,119]. And these researches are adopting the method of modifying the transport protocol itself.

To figure out the problem of fairness, We decided to approach it from a different perspective in the future. As we discussed in section 2.3, SDN concept brings us a new flexible network operation and application level control is possible. The standardized OpenFlow protocol provides Quality of Service (QoS) functionality called per-flow meters. Per-flow meters enable OpenFlow to implement QoS operations, such as rate-limiting per flow and can be combined with per-port queues to implement complex QoS frameworks.

Since our system is based on SDN, we can adopt QoS function to our system to realize fairness. There are two conditions that should be considered in real network. 1) Make sure the TCP streams of GridFTP do not affect other communications when GridFTP starts its data transfer. 2) Network conditions change dynamically; for example, the other users may start new TCP connections when the GridFTP is working. Therefore, it is necessary to guarantee the quality of the new network connection.

We design two scenarios to deal with the two conditions. For the first condition, we can limit the maximum rate of GridFTP's TCP streams before data transfer. As simulation experiments [118] shown, if we can control the bandwidth utilization less than 90% on the paths that GridFTP will use, it will not cause unfairness. For the condition 2, since dynamic QoS management of SDN is an effective method which was shown in several studies [120–122], the solution is further and dynamically limiting the bandwidth utilization of GridFTP transfer when OpenFlow controller detects other new TCP connections. Furthermore, the OpenFlow-compatible switches from different vendors also show different behaviors utilizing dynamic QoS management [123]; it is necessary to choose an appropriate switch to realize QoS mechanisms.

7.5 Determine the Parameter of Prediction Model on PRAGMA-ENT

In our current OpenFlow controller, to simplify the design, we used a common parameter a of a prediction model for all paths. Actually, for each network path, the performance could be improved by assigning different parameter a according to network path conditions. Therefore, we plan to adapt the parameter adjustment function to improve the OpenFlow controller and conduct further experiments.

In addition, we measured the parameter and network conditions manually before data transfer. Moreover, our proposed optimization of TCP Streams assignment method based on the accurate information of bandwidth and latency. It is also possible to implement a network monitoring to automatically realize the measurement required for the prediction model.

7.6 Applications in Other Areas

Our research aims to provide a high-speed data transfer system to the computational science research projects. Furthermore, other data transfer service such as data backup, Disaster Recovery (DR) and multimedia streaming can also benefit from our proposed multipath OpenFlow controller and prediction model of

optimal TCP streams.

In order to cope with any kind of service disruption from accidental interference, many enterprises prepare several data centers to back up data and its own DR system. Since the data generated daily is very huge, high-speed transfer service between the data centers is necessary especially under the process of DR. In this kind of case, our proposed system can support the high-speed data backup and DR by providing optimal multipath data transfer.

In addition, to support high-performance multimedia streaming, Content Delivery Network (CDN) technology is widely used for content delivery. CDN has a lot of geographically distributed servers which store the cached version of necessary content. This method provides a better user experience by minimizing the distance between the end-users and web server. However, when some new large-scale data (such as the application of 8k video streaming service) needs to be distributed between the servers of CDN, high-speed data transfer service is very important. Therefore, it is possible to apply our proposed system to the CND which can optimize the network utilization and achieve a better multimedia streaming service.

8 Conclusion

In this thesis, considering the need for the big data era, we focused on the high-speed transfer of large-scale data. Especially in data-intensive scientific projects, high-speed data transfer between sites is a necessary platform service. We surveyed the current techniques for high-speed data transfer. Regarding this topic, there are two main studies that are about transfer protocols and traffic engineering techniques.

We first explained various transfer protocols for high bandwidth utilization. After researching on many performance evaluations in which these techniques were compared, we chose the multiple TCP streams techniques for our research. There are also a lot of methods to realize multiple TCP streams transfer, however, since the application-level protocol is easy to deploy and utilize than transport layer for users, we decided to use the application layer to realize multiple TCP streams. Furthermore, in the computational science research fields, GridFTP was widely used and standardized by the Global Grid Forum. We therefore chose GridFTP as our method to realize data transfer with multiple TCP streams.

Regarding the traffic engineering techniques, since using multipath can aggregate more bandwidth, we introduced current routing techniques for utilizing multiple paths by dividing them into the source and Hop-by-Hop routing. However, all the current multipath routing methods utilize additional header/label or extended routing protocols and would cause additional overhead in both the control and data planes. Furthermore, these methods are not easy to deploy and lack complete central control. Hence, we adopted SDN technique to realize more flexible traffic engineering.

As a result, we proposed a system which combined GridFTP and SDN. We implemented a multipath OpenFlow controller based on OpenFlow to provide multipath to any application. To make GridFTP utilize our controller, we im-

plemented a Globus XIO that can communicate with the controller. Moreover, to optimize the data transfer of multiple TCP streams, we proposed a prediction model according to the network condition of each path. We also verified the effectiveness of our model in our virtual experimental environment.

To demonstrate the effectiveness of our proposed system, we built a virtual environment and a real global-scale environment and performed various experiments. The results demonstrate that our proposed system accelerates the data transfer of GridFTP and fully utilizes network resources. In the real global-scale experimental environment, the results show the practicality of our proposal and indicate that our proposed method pushes the performance to near the physical limitations of the hardware.

The main contribution of this thesis is proposing a system for high-speed data transfer. Our system utilizes a multipath controller providing multiple paths to the multiple TCP streams of GridFTP by using SDN technology based on OpenFlow. We also proposed an effective prediction model for optimal assignment of TCP streams. In terms of future work, we discussed possible issues when actually utilize our proposed system. We also provided some feasible methods to deal with these possible issues.

Acknowledgements

I would like to thank the following people for their wisdom, guidance, and support. Without whose help this work would never have been possible:

First and foremost, I would like to express a deep appreciation to my supervisor, Professor Hajimu Iida, who provides me a great opportunity to study at Laboratory for Software Design and Analysis. Not only he gave me helpful advices for my research, but also he motivated me to join business contests through which I could express my ideas beyond the academic field. It largely widened my perspectives in many ways, and through the given chances I was able to elevate my professional aptitude to a higher level that I became ready for the upcoming challenges.

I would like to thank Professor, Kazutoshi Fujikawa. As a member of my dissertation committee, his suggestions given in my Seminar presentation also benefits me with improving some critical areas of my research.

I would like to express my deepest appreciation to Associate Professor, Kohei Ichikawa. I met professor Ichikawa five years ago at the open campus of NAIST when I was studying for my master degree at Osaka Kyoiku University. I was attracted by his research direction and his passion for research. As such, I decided to continue my doctoral course under his guidance. Although I was not major in the field of networking during my year in Master Course, he accepted me and guided me to a successful direction in networking research. His continuous help and suggestions for my research allowed me to successfully accomplish the doctoral course.

Moreover, I would also like to express my gratitude to Associate Professor, Yasuhiro Watashiba. Professor Watashiba gave me valuable advices on writing research papers and conference presentations, through which I became much better than before in expressing my research work professionally in the academic field.

I would like to thank my thesis committee. Thank you so much for reviewing my thesis and for the insightful comments and suggestions that help me improve the overall quality of this thesis.

My sincere thanks also goes to Dr. Ryousei Takano and Dr. Takahiro Hirofuchi in National Institute of Advanced Industrial Science and Technology (AIST), who

provided me an opportunity to join their team an intern, and gave me access to the laboratory and research institutes.

Also I want to especially thank Professor Junichi Fujii and Professor Yuri Hasegawa in Osaka Kyoiku University. Professor Fujii was my supervisor during both my bachelor and master course. His guidance with passion helped me set up a strog foundation of knowledges in the field of Informmation Science. Furthermore, he also encouraged me to push further for my Doctoral degree. In addition, Professor Yuri Hasegawa was my Japanese language lecturer. Instead of only teaching me about the language itself, she helped me understand the culture, gave me tremendous advice and assistance for my daily life in Japan. With her generous help, I was able to pursue my academic achievement, and build my family in this wonderful country.

I wish to express my great thanks to MEXT and JASSO for supporting my research and life in Japan since I was an undergraduate student. I feel deeply honored to receive the scholarship from them and this honor strongly encouraged me to work harder.

Finally and most importantly, I would like to thank my beloved parents and wife. Their unconditional love and endless support make me have this gorgeous day. Thank you for letting me follow my dreams and happily supporting me to be whatever I want to be. I would like to dedicate this thesis to them as a small present of my gratitude for everything.

References

[1] Viktor Mayer-Schönberger and Kenneth Cukier. *Big Data: A Revolution that Will Transform How We Live, Work, and Think*. Eamon Dolan/-Mariner Books; Reprint edition, 4 2014.

[2] James Manyika. *The Internet of Things: Mapping the value beyond the hype*. McKinsey Global Institute, 2015.

[3] IBM. 10 key marketing trends for 2017. [Online]. Available: `https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN`, Accessed on: Dec 1, 2017, 2016.

[4] Eric Siegel. *Predictive analytics: The power to predict who will click, buy, lie, or die*. Wiley Hoboken (NJ), 2016.

[5] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23, 2000.

[6] Reagan Moore, Chaitanya Baru, Richard Marciano, Arcot Rajasekar, and Michael Wan. Data-intensive computing. *The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann*, pages 105–129, 1999.

[7] Mark Allman, Hans Kruse, and Shawn Ostermann. An application-level solution to TCP's satellite inefficiencies. In *Proceedings of the First International Workshop on Satellite-based Information Services (WOSBIS)*. Citeseer, 1996.

[8] Harimath Sivakumar, Stuart Bailey, and Robert L Grossman. Psockets: The case for application-level network striping for data intensive applica-

tions using high speed wide area networks. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 38–38. IEEE, 2000.

[9] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The Globus striped GridFTP framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society, 2005.

[10] John Bresnahan, Michael Link, Gaurav Khanna, Zulfikar Imani, Rajkumar Kettimuthu, and Ian Foster. Globus gridftp: what's new in 2007. In *Proceedings of the first international conference on Networks for grid applications*, page 19. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007.

[11] Sunand Tullimas, Thinh Nguyen, Rich Edgecomb, and Sen-ching Cheung. Multimedia streaming using multiple TCP connections. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 4(2):12, 2008.

[12] Andrea Baldini, Lorenzo De Carli, and Fulvio Risso. Increasing performances of TCP data transfers through multiple parallel connections. In *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*, pages 630–636. IEEE, 2009.

[13] Barkatullah Qureshi, Mohamed Othman, Shamala Subramaniam, and Nor Asila Wati. Qtcp: improving throughput performance evaluation with high-speed networks. *Arabian Journal for Science and Engineering*, 38(10):2663–2691, 2013.

[14] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, and Janardhan Iyengar. Architectural guidelines for multipath TCP development. In *Request for Comments (RFC) 6182*, 2011.

[15] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[16] Open Networking Foundation. [Online]. Available: https://www.opennetworking.org/, Accessed on: July 22, 2016.

[17] Serguei Chatrchyan, G Hmayakyan, V Khachatryan, AM Sirunyan, Wolfgang Adam, T Bauer, Thomas Bergauer, H Bergauer, M Dragicevic, Janos Erö, et al. The cms experiment at the cern lhc. 2008.

[18] Alex Abramovici, William E Althouse, Ronald WP Drever, Yekta Gürsel, Seiji Kawamura, Frederick J Raab, David Shoemaker, Lisa Sievers, Robert E Spero, Kip S Thorne, et al. Ligo: The laser interferometer gravitational-wave observatory. *science*, 256(5055):325–333, 1992.

[19] Georges Aad, E Abat, J Abdallah, AA Abdelalim, A Abdesselam, O Abdinov, BA Abi, M Abolins, H Abramowicz, E Acerbi, et al. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(8):S08003–S08003, 2008.

[20] Worldwide LHC Computing Grid. [Online]. Available: http://wlcg.web.cern.ch/, Accessed on: Dec 1, 2017.

[21] Energy Sciences Network. [Online]. Available: https://www.es.net/, Accessed on: Dec 1, 2017.

[22] GÉANT. [Online]. Available: https://www.geant.org/Networks, Accessed on: Dec 1, 2017.

[23] China Education and Research Network. [Online]. Available: http://www.edu.cn/HomePage/english/cernet/, Accessed on: Dec 1, 2017.

[24] Takashi Kurimoto, Shigeo Urushidani, Hiroshi Yamada, Kenjiro Yamanaka, Motonori Nakamura, Shunji Abe, Kensuke Fukuda, Michihiro Koibuchi, Hiroki Takakura, Shigeki Yamada, et al. Sinet5: A low-latency and high-bandwidth backbone network for sdn/nfv era. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–7. IEEE, 2017.

[25] Jon Postel. Transmission control protocol. 1981.

[26] Kun-chan Lan and John Heidemann. A measurement study of correlations of internet flow characteristics. *Computer Networks*, 50(1):46–62, 2006.

[27] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. Tcp revisited: a fresh look at tcp in the wild. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 76–89. ACM, 2009.

[28] Haiqing Jiang, Yaogong Wang, Kyunghan Lee, and Injong Rhee. Tackling bufferbloat in 3g/4g networks. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 329–342. ACM, 2012.

[29] W Richard Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. 1997.

[30] Mark Allman, Vern Paxson, and Ethan Blanton. Tcp congestion control. Technical report, 2009.

[31] Jon Crowcroft and Philippe Oechslin. Differentiated end-to-end Internet services using a weighted proportional fair sharing TCP. *ACM SIGCOMM Computer Communication Review*, 28(3):53–69, 1998.

[32] Mario Gerla, Medy Y Sanadidi, Ren Wang, Andrea Zanella, Claudio Casetti, and Saverio Mascolo. TCP Westwood: Congestion window control using bandwidth estimation. In *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, volume 3, pages 1698–1702. IEEE, 2001.

[33] Hung-Yun Hsieh and Raghupathy Sivakumar. ptcp: An end-to-end transport layer protocol for striped connections. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 24–33. IEEE, 2002.

[34] Sally Floyd. Highspeed tcp for large congestion windows. 2003.

[35] Tom Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM computer communication Review*, 33(2):83–91, 2003.

[36] Douglas Leith and Robert Shorten. H-tcp: Tcp for high-speed and long-distance networks. In *Proceedings of PFLDnet*, volume 2004, 2004.

[37] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524. IEEE, 2004.

[38] Sumitha Bhandarkar, Saurabh Jain, and AL Narasimha Reddy. Improving tcp performance in high bandwidth high rtt links using layered congestion control. *PFLDNet'05*, 2005.

[39] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. Fast tcp: motivation, architecture, algorithms, performance. *IEEE/ACM transactions on Networking*, 14(6):1246–1259, 2006.

[40] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[41] Jon Postel. User datagram protocol. Technical report, 1980.

[42] Yunhong Gu and Robert Grossman. Sabul: A transport protocol for grid computing. *Journal of Grid Computing*, 1(4):377–386, 2003.

[43] Malathi Veeraraghavan, Xuan Zheng, Hyuk Lee, M Gardner, and Wuchun Feng. Cheetah: Circuit-switched high-speed end-to-end transport architecture. In *Proceedings of SPIE*, volume 5285, pages 214–225, 2003.

[44] Qishi Wu and Nageswara SV Rao. Protocol for high-speed data transport over dedicated channels. In *Third International Workshop on Protocols for Long-Distance Networks (PFLDnet 2005), Lyon, France*, 2005.

[45] Xuan Zheng, Anant Padmanath Mudambi, and Malathi Veeraraghavan. Frtp: Fixed rate transport protocol–a modified version of sabul for end-to-end circuits. In *Proc. of Broadnets*, 2004.

[46] Chaoyue Xiong, Jason Leigh, Eric He, Venkatram Vishwanath, Tadao Murata, Luc Renambot, and T DeFanti. Lambdastream–a data transport protocol for streaming network-intensive applications over photonic networks. In *Third International Workshop on Protocols for Long-Distance Networks (PFLDnet 2005), Lyon, France*, volume 11, 2005.

[47] Yunhong Gu and Robert L Grossman. UDT: UDP-based data transfer for high-speed wide area networks. *Computer Networks*, 51(7):1777–1799, 2007.

[48] Mohamed A Alrshah and Mohamed Othman. Test-bed based comparison of single and parallel TCP and the impact of parallelism on throughput and fairness in heterogenous networks. In *Computer Technology and Development, 2009. ICCTD'09. International Conference on*, volume 1, pages 332–335. IEEE, 2009.

[49] Mohamed A Alrshah and Mohamed Othman. Performance evaluation of parallel TCP, and its impact on bandwidth utilization and fairness in High-BDP networks based on test-bed. In *Communications (MICC), 2013 IEEE Malaysia International Conference on*, pages 23–28. IEEE, 2013.

[50] Lyndon Ong. An introduction to the stream control transmission protocol (sctp). 2002.

[51] Yohei Hasegawa, Ichiro Yamaguchi, Takayuki Hama, Hideyuki Shimonishi, and Tutomu Murase. Improved data distribution for multipath tcp communication. In *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, volume 1, pages 5–pp. IEEE, 2005.

[52] Younghak Hwang, Brownson O Obele, and Hyuk Lim. Multipath transport protocol for heterogeneous multi-homing networks. In *Proceedings of the ACM CoNEXT Student Workshop*, page 5. ACM, 2010.

[53] Bhargava K Kancherla, Ganesh M Narayan, and K Gopinath. Performance evaluation of multiple tcp connections in iscsi. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on*, pages 239–244. IEEE, 2007.

[54] Ronald van der Pol, Michael Bredel, Artur Barczyk, Benno Overeinder, Niels van Adrichem, and Fernando Kuipers. Experiences with mptcp in an intercontinental openflow network. In *Proceedings of the 29th TERENA Network Conference (TNC2013)*, 2013.

[55] Jon Postel and Joyce Reynolds. File transfer protocol. October 1985.

[56] Marc Horowitz and Steve Lunt. Ftp security extensions. Technical report, October 1997.

[57] Paul Hethmon and Robert Elz. Feature negotiation mechanism for the file transfer protocol. August 1998.

[58] Charlie Catlett. Standards for grid computing: Global grid forum. *Journal of Grid Computing*, 1(1):3–7, 2003.

[59] William Allcock. Gridftp: Protocol extensions to ftp for the grid. *OGF Document Series GFD.20*, April 2003.

[60] Ian Foster and Carl Kesselman. The globus project: A status report. *Future Generation Computer Systems*, 15(5):607–621, 1999.

[61] Yunhong Gu. *UDT: a high performance data transport protocol*. University of Illinois at Chicago, 2005.

[62] John Bresnahan, Michael Link, Rajkumar Kettimuthu, and Ian Foster. Udt as an alternative transport protocol for gridftp. In *International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, pages 21–22, 2009.

[63] Se-young Yu, Nevil Brownlee, and Aniket Mahanti. Comparative performance analysis of high-speed transfer protocols for big data. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 292–295. IEEE, 2013.

[64] Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (bgp-4). Technical report, 2005.

[65] Charles L Hedrick. Routing information protocol. Technical report, 1988.

[66] John Moy. Ospf version 2. 1998.

[67] Curtis Villamizar. Ospf optimized multipath (ospf-omp). *Work in Progress*, 1999.

[68] Dapeng Zhu, Mark Gritter, and David R Cheriton. Feedback based routing. *ACM SIGCOMM Computer Communication Review*, 33(1):71–76, 2003.

[69] H Tahilramani Kaur, Shivkumar Kalyanaraman, Andreas Weiss, Shifalika Kanwar, and Ayesha Gandhi. Bananas: An evolutionary framework for explicit and multipath routing in the internet. *ACM SIGCOMM Computer Communication Review*, 33(4):277–288, 2003.

[70] P Krishna Gummadi, Harsha V Madhyastha, Steven D Gribble, Henry M Levy, David Wetherall, et al. Improving the reliability of internet paths with one-hop source routing. In *OSDI*, volume 4, pages 13–13, 2004.

[71] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 159–170. ACM, 2006.

[72] Xiaowei Yang, David Clark, and Arthur W Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Transactions on Networking (ToN)*, 15(4):775–788, 2007.

[73] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 27–38. ACM, 2008.

[74] P Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. *ACM SIGCOMM Computer Communication Review*, 39(4):111–122, 2009.

[75] Donghong Qin, Jaihai Yang, Zhuolin Liu, Hui Wang, Bin Zhang, and Wei Zhang. Amir: Another multipath interdomain routing. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 581–588. IEEE, 2012.

[76] Daniel O Awduche and Johnson Agogbua. Requirements for traffic engineering over mpls. 1999.

[77] Yongho Seok, Youngseok Lee, Yanghee Choi, and Changhoon Kim. Dynamic constrained multipath routing for mpls networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 348–353. IEEE, 2001.

[78] Jiayue He and Jennifer Rexford. Toward internet-wide multipath routing. *IEEE network*, 22(2), 2008.

[79] Christian E Hopps. Analysis of an equal-cost multi-path algorithm. 2000.

[80] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing OSPF weights. In *INFOCOM 2000. Nineteenth annual joint conference of the IEEE computer and communications societies.*, volume 2, pages 519–528. IEEE, 2000.

[81] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for internet load balancing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 332–341. IEEE, 2000.

[82] Feng Wang and Lixin Gao. Path diversity aware interdomain routing. In *INFOCOM 2009, IEEE*, pages 307–315. IEEE, 2009.

[83] Iljitsch Van Beijnum, Jon Crowcroft, Francisco Valera, and Marcelo Bagnulo. Loop-freeness in multipath bgp through propagating the longest path. In *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*, pages 1–6. IEEE, 2009.

[84] Nate Kushman, Srikanth Kandula, Dina Katabi, and Bruce M Maggs. R-bgp: Staying connected in a connected world. USENIX, 2007.

[85] Igor Ganichev, Bin Dai, P Godfrey, and Scott Shenker. Yamr: Yet another multipath routing protocol. *ACM SIGCOMM Computer Communication Review*, 40(5):13–19, 2010.

[86] Hideyuki Shimonishi, Yasuhito Takamiya, Yasunobu Chiba, Kazushi Sugyo, Youichi Hatano, Kentaro Sonoda, Kazuya Suzuki, Daisuke Kotani, and Ippei Akiyoshi. Programmable network using OpenFlow for network researches and experiments. In *Proc. 6th International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2012)*, pages 164–171, 2012.

[87] POX. [Online]. Available: `https://github.com/noxrepo/pox/`, Accessed on: July 22, 2016.

[88] NOX open controller. [Online]. Available: `http://www.noxrepo.org/`, Accessed on: July 22, 2016.

[89] Floodlight OpenFlow controller -Project Floodlight. [Online]. Available: `http://www.projectfloodlight.org/floodlight/`, Accessed on: July 22, 2016.

[90] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.

[91] The OpenDaylightPlatform. [Online]. Available: `https://www.opendaylight.org/`, Accessed on: July 22, 2016.

[92] Ryu SDN framework. [Online]. Available: `http://osrg.github.io/ryu/`, Accessed on: July 22, 2016.

[93] Bing Wang, Wei Wei, Jim Kurose, Don Towsley, Krishna R Pattipati, Zheng Guo, and Zheng Peng. Application-layer multipath data transfer via TCP: schemes and performance tradeoffs. *Performance Evaluation*, 64(9):965–977, 2007.

[94] Jinyu Zhang, Yongzhe Gui, Cheng Liu, and Xiaoming Li. To improve throughput via multi-pathing and Parallel TCP on each path. In *ChinaGrid Annual Conference, 2009. ChinaGrid'09. Fourth*, pages 16–21. IEEE, 2009.

[95] Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry L Peterson, and Randolph Y Wang. A Transport Layer Approach for Improving End-to-End

Performance and Robustness Using Redundant Paths. In *USENIX Annual Technical Conference, General Track*, pages 99–112, 2004.

[96] Ezra Kissel, Martin Swany, and Aaron Brown. Improving GridFTP performance using the Phoebus session layer. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 34. ACM, 2009.

[97] Dan Gunter, Rajkumar Kettimuthu, Ezra Kissel, Martin Swany, Jun Yi, and Jason Zurawski. Exploiting Network Parallelism for Improving Data Transfer Performance. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1600–1606. IEEE, 2012.

[98] Yoshihiko Kanaumi, Shu-ichi Saito, Eiji Kawai, Shuji Ishii, Kazumasa Kobayashi, and Shinji Shimojo. RISE: A Wide-Area Hybrid OpenFlow Network Testbed. *Ieice Transactions on Communications*, 96(1):108–118, 2013.

[99] Shuji Ishii, Eiji Kawai, Yoshihiko Kanaumi, Shu-ichi Saito, Tomoaki Takata, Kazumasa Kobayashi, and Shinji Shimojo. A study on designing OpenFlow controller RISE 3.0. In *Networks (ICON), 2013 19th IEEE International Conference on*, pages 1–5. IEEE, 2013.

[100] NEC Corporation. Routing switch, Trema Apps. [Online]. Available: `https://github.com/trema/apps/tree/master/routing_switch`, Accessed on: July 22, 2016.

[101] Trema App. [Online]. Available: `https://github.com/trema/apps/`, Accessed on: July 22, 2016.

[102] Phillip Merrick, Stewart Allen, and Joseph Lapp. Xml remote procedure call (xml-rpc), 2006. US Patent 7,028,312.

[103] Thomas J Hacker, Brian D Athey, and Brian Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS02)*, pages 434–443. IEEE, 2002.

[104] Dong Lu, Yi Qiao, Peter A Dinda, and Fabian E Bustamante. Modeling and taming parallel tcp on the wide area network. In *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2005.

[105] Takeshi Ito, Hiroyuki Ohsaki, and Makoto Imase. On parameter tuning of data transfer protocol gridftp for wide-area grid computing. In *2nd International Conference on Broadband Networks, 2005.*, pages 1338–1344. IEEE, 2005.

[106] Bert Hubert et al. Linux advanced routing & traffic control HOWTO. `http://www.lartc.org/howto/.`, 2009.

[107] Open vSwitch. [Online]. Available: `http://openvswitch.org`, Accessed on: July 22, 2016.

[108] Dino Farinacci, P Traina, Stan Hanks, and T Li. Generic routing encapsulation (gre). 1994.

[109] Kohei Ichikawa, Mauricio Tsugawa, Jason Haga, Hiroaki Yamanaka, Te-Lung Liu, Yoshiyuki Kido, Pongsakorn U-Chupala, Che Huang, Chawanat Nakasan, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Susumu Date, Shinji Shimojo, Philip Papadopoulos, and Jose Fortes. PRAGMA-ENT: Exposing SDN concepts to domain scientists in the pacific rim. In *PRAGMA Workshop on International Clouds for Data Science (PRAGMA-ICDS 2015)*, 2015.

[110] Kohei Ichikawa, Pongsakorn U-chupala, Che Huang, Chawanat Nakasan, Te-Lung Liu, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Jason Haga, Hiroaki Yamanaka, Eiji Kawai, et al. PRAGMA-ENT: An International SDN testbed for cyberinfrastructure in the Pacific Rim. *Concurrency and Computation: Practice and Experience*, 29(13), 2017.

[111] Mininet: An instant virtual network on your laptop (or other PC). [Online]. Available: `http://www.mininet.org/`, Accessed on: July 22, 2017.

[112] Alexander Shalimov, Dmitry Zuikov, Daria Zimarina, Vasily Pashkov, and Ruslan Smeliansky. Advanced study of SDN/OpenFlow controllers. In

Proceedings of the 9th central & eastern european software engineering conference in russia, page 1. ACM, 2013.

[113] Farnam Jahanian, Craig Labovitz, and Abha Ahuja. Experimental study of internet stability and wide-area backbone failures. *U. of Michigan, Tech. Rep. CSE-TR-382-98*, 1998.

[114] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 3–16. ACM, 2002.

[115] Athina Markopoulou, Gianluca Iannaccone, Supratik Bhattacharyya, Chen-Nee Chuah, and Christophe Diot. Characterization of failures in an ip backbone. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2307–2317. IEEE, 2004.

[116] Nick Feamster, David G Andersen, Hari Balakrishnan, and M Frans Kaashoek. Measuring the effects of internet path faults on reactive routing. In *ACM SIGMETRICS Performance Evaluation Review*, volume 31, pages 126–137. ACM, 2003.

[117] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. *ACM SIGCOMM Computer Communication Review*, 30(4):175–187, 2000.

[118] Thomas J Hacker, Brian D Noble, and Brian D Athey. The effects of systemic packet loss on aggregate tcp flows. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 7–7. IEEE, 2002.

[119] Thomas J Hacker, Brian D Noble, and Brian D Athey. Improving throughput and maintaining fairness using parallel TCP. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2480–2489. IEEE, 2004.

[120] Panagiotis Georgopoulos, Yehia Elkhatib, Matthew Broadbent, Mu Mu, and Nicholas Race. Towards network-wide qoe fairness using openflow-

assisted adaptive video streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pages 15–20. ACM, 2013.

[121] Thomas Zinner, Michael Jarschel, Andreas Blenk, Florian Wamser, and Wolfgang Kellerer. Dynamic application-aware resource management using software-defined networking: Implementation prospects and challenges. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–6. IEEE, 2014.

[122] Wonho Kim, Puneet Sharma, Jeongkeun Lee, Sujata Banerjee, Jean Tourrilhes, Sung-Ju Lee, and Praveen Yalagandula. Automated and scalable qos control for network convergence. *INM/WREN*, 10(1):1–6, 2010.

[123] Raphael Durner, Andreas Blenk, and Wolfgang Kellerer. Performance study of dynamic qos management for openflow-enabled sdn switches. In *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*, pages 177–182. IEEE, 2015.

Publication List

Refereed

- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "A Multipath OpenFlow Controller for Multiple TCP Stream Applications," IPSJ Transactions on Advanced Computing System, 2017.
- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "A Multipath OpenFlow Controller for GridFTP," The 1st. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming, April 2017.
- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "An SDN-based Multipath Gridftp for High-Speed Data Transfer," In 36th IEEE International Conference on Distributed Computing Systems, pages 763-764, June 2016.
- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "A Multipath Controller for Accelerating GridFTP Transfer Over SDN," In 11th IEEE International Conference on eScience, pages 439-447, September 2015.

Non-Refereed, Oral Presentation

- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "An Optimal Multipath Assignment Technique for OpenFLow Network," PRAGMA 33 Workshop, Poster, October 2017.
- **Che Huang**, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "A Multipath Controller for Accelerating GridFTP Transfer Over SDN," PRAGMA 28 Workshop, Poster, April 2015.

Other Related Publication

- Kohei Ichikawa, Pongsakorn U-chupala, **Che Huang**, Chawanat Nakasan, Te-Lung Liu, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Jason Haga, Hiroaki Yamanaka, Eiji Kawai, Yoshiyuki Kido, Susumu Date, Shinji Shimojo, Philip Papadopoulos, Mauricio Tsugawa, Matthew Collins, Kyuho Jeong, Renato Figueiredo, and Jose Fortes, "Pragma-Ent: an International SDN Testbed for a Cyberinfrastructure in the Pacific Rim," Concurrency And Computation: Practice And Experience, e4138 March 2017.

- Kohei Ichikawa, Mauricio Tsugawa, Jason Haga, Hiroaki Yamanaka, Te-Lung Liu, Yoshiyuki Kido, Pongsakorn U-Chupala, **Che Huang**, Chawanat Nakasan, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Susumu Date, Shinji Shimojo, Philip Papadopoulos, and Jose Fortes, "Pragma-Ent: exposing SDN concepts to domain scientists in the Pacific Rim," In PRAGMA Workshop on International Clouds for Data Science, October 2015.

Awards

- Best Poster Award

  Che Huang, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "An Optimal Multipath Assignment Technique for OpenFLow Network," PRAGMA 33 Workshop, Poster, October 2017.

- Best Research Award

  Che Huang, Chawanat Nakasan, Kohei Ichikawa, and Hajimu Iida, "Best Research Award," The 1st. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming, April 2017.

- Best Paper Award

  Kohei Ichikawa, Mauricio Tsugawa, Jason Haga, Hiroaki Yamanaka, Te-Lung Liu, Yoshiyuki Kido, Pongsakorn U-Chupala, **Che Huang**, Chawanat Nakasan, Jo-Yu Chang, Li-Chi Ku, Whey-Fone Tsai, Susumu Date, Shinji Shimojo, Philip Papadopoulos, and Jose Fortes, "Pragma-Ent: exposing SDN concepts to domain scientists in the Pacific Rim," In PRAGMA Workshop on International Clouds for Data Science, October 2015.