NAIST-IS-DD1561029

Doctoral Dissertation

Improvement of Multipath TCP Performance using Software-Defined Network

Chawanat Nakasan

March 8, 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

Chawanat Nakasan

Thesis Committee:

Professor Hajimu Iida	(Supervisor)
Professor Kazutoshi Fujikawa	(Co-supervisor)
Associate Professor Kohei Ichikawa	(Co-supervisor)
Associate Professor Yasuhiro Watashiba	(Co-Supervisor)
Assistant Professor Putchong Uthayopas	(Kasetsart University)

Improvement of Multipath TCP Performance using Software-Defined Network *

Chawanat Nakasan

Abstract

Current distributed systems require a large and increasing amount of network resources and greatly benefit from a larger bandwidth. Network environments, such as in data centers and across wide-area networks, usually provide multiple paths for each host to mitigate possible network failures. However, even with multiple paths to utilize, only one path is used per connection, which severely limits the maximum throughput. Additionally, there is no comprehensive multipathing solution as of the start of this research. Even by using transport-layer multipathing protocol such as MPTCP, it is still incomplete because this protocol is not in control of network routing. By providing a multipathing-aware routing system it is possible to increase useable bandwidth between two hosts and provide a more stable connection, both of which are crucial requirements of many distributed systems.

There are previous work that has been conducted on routing in high-performance networks like this, but so far they are mostly concerned about creating general-purpose bandwidth maximization algorithms in routing, without concern about optimizing for specific cases such as MPTCP and software-defined networks. On the other side, recent research on "multipath routing" concerns more about wireless and ad hoc communications, which are not the case for data center networks or wide-area networks. By thoroughly studying and carefully combining multipath routing theories and software-defined network, it is possible to improve the performance of distributed systems.

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, NAIST-IS-DD1561029, March 8, 2018.

In this dissertation, a better routing algorithm that is tailored for MPTCP is explored, along with other similar algorithms that promise similar qualities. It is expected that this algorithm will, in terms of total throughput between a single pair of hosts, perform as well as simple shortest-path algorithms under the conditions of a spanning-tree network, or general networks without the use of MPTCP, while performing better when both multiple paths and MPTCP are available. This work is expected to result in a better performance for distributed systems, and outlines the possible ways to increase available throughput and increase the performance.

Keywords:

computer networks, distributed storage, software-defined network, OpenFlow, Multipath TCP, multipath networking, multipath routing

Contents

List of Figures

1 Introduction

1.1 Background: The Demands of Networked Systems

Networked systems, such as database, computation, and storage systems, have become more prominent in the society as infrastructure supporting various applications. The increase in usage, complexity, availability, adoption, among many factors, have led to increasing demands on the network. Furthermore, the widespread adoption of cloud computing adds a new dimension to the usage of network resources. With abstraction concepts such as infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS), many services were widely created, leveraged, and utilized. Coupled with user-generated nature of modern Internet, the demands on the networks have further increased, as practically anyone can create a website, service, or even a whole virtualized data center in a matter of minutes.

1.2 Keeping Up with the Data Demands

In a data center network (DCN), it is possible to invest in equipment so that the system can handle more traffic, work quicker, and are more stable. On the other hand, wide-area networks (WANs) which are more prone to latency can greatly benefit from a more direct route to their destinations. This is why Internet data centers (IDCs) are born in the first place – to provide a direct Internet connection to the server and benefiting from proximity to the Internet to reduce their latency.

Systems in WANs can benefit further from practices like *multi-homing* which connects a system to the Internet through multiple gateways. Another concept, *multi-site*, refers to the practice of distributing the system to multiple geographic

locations. These practices have many benefits including locality, capacity, and redundancy. When these two concepts are used together, the sites of the networked system can be connected through wide-area network (WAN) by multiple paths. Similarly, DCNs can also provide multiple paths to each server unit by installing additional network elements, such as network interfaces and switches, to increase the number of connections and benefit from the improved reliability as well as potential bandwidth, something which would be discussed further in Chapter 2. In fact, some motherboards are designed with multiple network interface cards (NICs) just for this purpose.

While new, more powerful equipment could be acquired to improve both the bandwidth and latency, modifying the networking mechanisms or protocols that run within them is not very easy due to the distributed nature of networking. Being protocols, they inherently require an agreement between multiple parties. It might be possible for proprietary protocols to be developed and tested inside a single organization, but it is much harder to gain the public acceptance needed for the protocols to successfully interoperate between machines, and also between autonomous systems. Any changes in networking must also take into account backward- and forward-compatibility, as the older devices may simply be unable to update to the new firmware necessary to run the newer protocols or operating systems that implement them.

While it is hopeful that "improving" the bandwidth would be a direct solution to bandwidth-dependent applications like networked systems, the situation of the networking field itself is not an easy affair. There are many reasons that lead to these difficulties, each a problem point of their own.

The first reason is the cost. Without naming names or numbering numbers, it is a well-known fact that network adapters and equipment that boast higher bandwidth, lower latency, faster processing rate, generally more stable, or otherwise "better", can be leagues more expensive than commodity devices used by the general public. There is also additional cost of actual infrastructure, such as renting out a better-connected data center, or constructing a private one for use in own company. For individual researchers, it might be difficult to pull the strings to gain access to these coveted hardware due to various reasons, be it technical, financial, or otherwise.

The second reason is the important barrier in networking known as the Shannon Limit [39] which can be plainly explained that no matter the modulation technique, there is only so much data that can be sent at a time without suffering from errors. It is suggested that as this limit is approached, a "trade-off between reach, bandwidth, and spectral efficiency" must be considered, and one way to overcome this problem is to use multiple network paths to provide more data capacity without hitting the Shannon Limit [34].

For the third reason, all networks, including local loopback, have latency. To put this into perspective, the distance from Tokyo to San Diego, approximately 9,000 km, would take light, its speed approximated at 3×10^8, 30 ms to reach the destination. Since communication would be done over a wire, the actual speed will be even lower because the propagation speed in a wire is lower than the speed of light itself. This constitutes the *propagation delay* of the often-quoted delay equation, which also consists of *transmission delay*, *processing delay*, and *queuing delay*. This means there is always some unavoidable latency in the network, which is only compounded by the time it takes for the network equipment to process and send or receive the data. While latency has minimal effect in DCNs and long-lived network transmissions such as streaming and large data chunks, it is devastating for response-demanding applications (such as interactive applications, including command shells, the Web, and games) and WANs.

Another important reason to be listed as the fourth here is the congestion control. The network cannot send all the data at once, especially with multiple users contending for resources. Congestion control mechanisms were created to throttle and slow down when necessary to avoid introducing errors in the network. However, most congestion control mechanisms in modern and general use use the round-trip time, which is a form of latency, as a parameter. Generally, the larger the latency, the less responsive the network will be towards congestion. This, in turn, causes reduction in performance as the end hosts cannot properly respond to the changes in network conditions in time, leading to packet losses or bandwidth under-utilization.

However, even if we tackle all the problem points of bandwidth, there is still an issue of system reliability. The network has always been capable of being designed with multiple paths between any pair of nodes, but it has not been widely utilized

as such because only one network path is used at a time. The traditional design of the network model itself, where one application-layer socket corresponds to only one transport-layer stream [10], which in turn corresponds to a fixed pair of network-layer and link-layer endpoints [41], permits only one path to be used per one connection. The path can be split or switched in the network layer, but it does not address the root at the endpoints of the connections. Classical routing mechanisms usually work in a way that results in only one route being used for each pair of source and destination hosts. This means the network resources (i.e. bandwidth) of multi-site multi-homed distributed systems would not be fully utilized if only one application socket is used to connect between multiple sites, as only one route would be used between the two sites.

A networking concept known as *multipathing* was developed to allow a host or network to utilize multiple paths at the same time. Multipathing has many benefits, including increasing maximum bandwidth, balancing network load, or providing redundancy. Many multipathing solutions exist, with their own set of benefits and problems.

Multipathing exists in many forms, and only some are actually useful in multi-site multi-homed systems that are connected through wide-area networks. By examining the TCP/IP model [41], we can find multiple places to perform multipathing, from application layer all the way down to physical layer, and many options were introduced. [10, 37, 23, 7, 13, 35, 12, 1, 2, 4, 44, 20] Among these, transport-layer multipathing protocol known as Multipath TCP (MPTCP) [12] stands out the most as it is both backwards-compatible with legacy applications and networks, and being a transport-layer protocol, it can regulate the flow of data to respond to congestion and therefore works well in unequal or unstable networks.

While useful, MPTCP is far from being the perfect solution to multi-site multi-homed systems on its own. Since MPTCP works on transport layer, it now suffers from the lack of control over which path(s) it should actually take and the network layer does not have any knowledge about MPTCP being used. As a result, it is possible that, in a shared infrastructure, the network would actually route the multiple paths in MPTCP through the same network route. This means the bandwidth in that same network route would have to be shared, eliminating the

benefits of MPTCP.

Software-defined networking or *SDN* can be utilized to rapidly test and implement new networking protocols and concepts with minimal cost.

The role of customizable routing suitable for fine-tuning MPTCP routing would be easily filled in by OpenFlow [26], an SDN protocol. OpenFlow allows a programmer to control various aspects of switching and routing in a network, by issuing commands from a single OpenFlow Controller to the OpenFlow Switches in the network. This centralized design gives a lot of power to the controller.

1.3 Improvement Factors for Bandwitdh Requirements of Distributed Systems

While there are many factors that decide what is considered "good" or "high-performance" for a network, and interpretations of different researchers and industries may differ, *bandwidth*, *latency*, *packet loss*, and *jitter* are considered to be among the most important metrics networks. However, with respect to the flexibility of the bandwidth problem and the emergence of data-intensive networking, it is a good idea to focus on the bandwidth aspect of the problem, which turns this into the objectives of this research.

The next point of consideration is how would OpenFlow and MPTCP be able to support bandwidth-intensive networking like this. The answer looks easy and the path looks logical, that it is possible to use MPTCP to split the network communication into multiple smaller subflows and manage them through multipath congestion control mechanisms, while OpenFlow manages the routing of those subflows.

1.4 Motivation and Goals

By looking at the problem in this way, we consider that creating a simple and efficient OpenFlow Controller that splits and distributes MPTCP traffic through the network would increase bandwidth utilization of MPTCP in the network, reducing waste in the form of underutilized bandwidth. To be specific, we are particularly

targeting wide-area networks because bandwidth in wide-area network is more limited, so greater efficiency means a lot more in this kind of network.

First and foremost, in order to preserve compatibility and allow our solution to be more easily deployed or used with existing systems, we also have a specific goal to not require changes in applications. This means it is most likely not possible to use an application-layer multipathing technique. Our solution should not break things that already work by introducing incompatibility.

Next, we focus on use of multiple paths and proportional increase of bandwidth. Applications should be able to use multiple paths automatically and benefit from the increased bandwidth when used in a system that we introduced.

Furthermore, we plan to find ways to improve path selection. There are many strategies to do so, but we will introduce one that can almost guarantee that not all flows will use the same route while being simple in itself.

Finally, we aim to adapt our solution and further refine upon the multipath routing strategy to find a topological routing strategy that works best in specific use cases.

1.4.1 Target and Use Cases

This work is primarily targeted at large-scale, multi-homed multi-site systems connected together using OpenFlow. Multi-homed multi-site systems include distributed storage, database, content delivery network (CDN), high-performance computing (HPC), or other critical systems with disaster recovery sites, which are usually located far away from the main site. For the purposes of this work, the experiments will use distributed storage as the subject. Apart from multi-homed multi-site WANs, a DCN environment with minimal latency and equal path cost will also be considered. The consideration of both WANs and DCNs means that network-layer multipathing techniques might not be applicable, further reinforcing the position of MPTCP as the multipath network protocol of choice in this work.

We specifically turn towards OpenFlow as an SDN protocol because it is a well-established standard with a long and established history of development and support from its developers, yet evolving quickly with active research and can be expected to remain relevant for foreseeable future.

1.5 Outline of this Dissertation

The rest of this dissertation is organized as follows. Chapter 2 describes background and related work, including the concepts of distributed storage, software-defined network, multipath networking in general, as well as multipath routing theories and algorithms. Chapter 3 discusses my early steps on the evaluation of Multipath TCP on an OpenFlow network. Chapter 4 revisits my previous work on the simple multipath OpenFlow controller (smoc) that I have implemented, with additional analyses and benchmarks. Chapter 5 talks about the analysis of how distributed storage systems communicate, which is a key point in understanding how bandwidth-intensive applications like them can be improved. Chapter 6 describes a collection of methods I have implemented to improve the routing algorithm using topological techniques, as well as relevant evaluations. Chapter 7 discusses the importance and impact of this work in its entirety, and the dissertation is finally wrapped up in Chapter 8.

2 Background

2.1 Software-defined Network

OpenFlow is an SDN protocol which allows network traffic control and management from the centralized OpenFlow Controller instead of distributing such control to each network element (switches, routers, etc.). By centralizing network control at the controller, the network elements can be programmed to add or remove any switching or routing rules in its *flow table*. Figure 2.1 shows a sample complete OpenFlow network, consisting of switches that are connected to the controller, making up the control plane, while ordinary switch-switch and switch-host connections make up the data plane. Since the controller has all the power to modify and influence the network, OpenFlow can be a very powerful network research and development tool.

While OpenFlow provides programming flexibility to the network and allows many concepts such as QoS or traffic engineering to be realized, it cannot modify communication pattern between the end hosts themselves. In traditional TCP/IP protocol suite, only one route or path is used per connection. This limits the

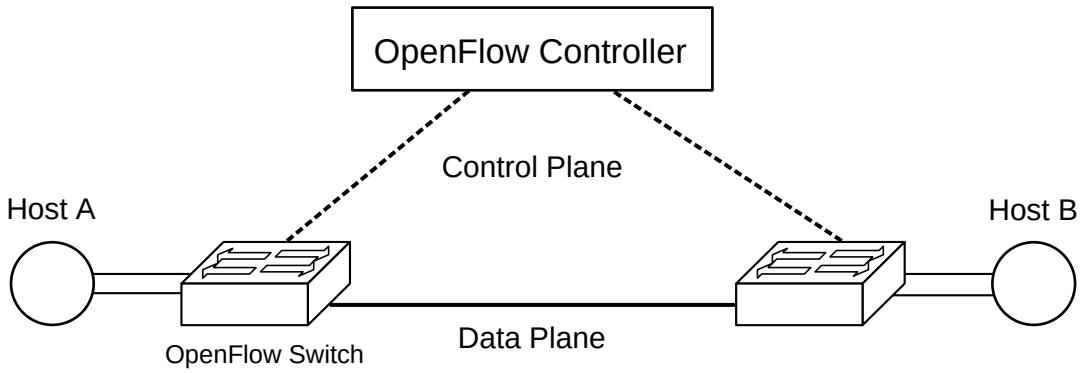


Figure 2.1: A simple image of OpenFlow, displaying the essential components.

maximum bandwidth of the entire connection to that of the segment with the least bandwidth, and this cannot be overcome by simply adopting OpenFlow.

2.2 Multipathing

One early mention of the term *multipath* is in RFC 2991, where the term is first mentioned in the phrase *Equal-Cost Multipath* (ECMP) routing, and explained that it is used in routing protocols such as Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (ISIS), along with other implementations such as in Routing Information Protocol (RIP) [44]. This means if, from one router's perspective, there are multiple paths to the destination with equal cost (which can be based on bandwidth, latency, or other metrics), then the router will use both of them. Therefore, the term multipathing in this case takes on the meaning of routing a connection or data stream between a single pair of hosts through multiple routes or paths. Since many other works in the field also use the same term to refer to this concept even though an explicit definition is not given [37, 4, 35], we will use this as the definition for *multipathing* in this thesis.

Since the term is loosely defined and there are many techniques or network protocols that achieve multipathing effect, we will begin by explaining them by layer, and highlight their differences.

2.2.1 Multipathing in Network Layer

RFCs 2991 and 2992 [44, 20] mention ECMP as a multipathing technique that works at the network layer. By using ECMP, routers will load-balance among the paths which have equal cost with respect to the destination. However, ECMP routing is performed at network layer and is problematic to TCP, because TCP is not aware of the presence of ECMP operations. Some ECMP implementations scatter packets in a round-robin fashion, making packets prone to arriving out-of-order which will prompt TCP to use retransmission mechanisms when multiple consecutive ACKs are received, decreasing network performance [10]. Another flaw of ECMP is the fact that it is designed for *equal*-cost networks and therefore will not exhibit multipathing when path costs are not equal, which is a common

occurrence in wide-area networks due to variation in bandwidth and latency. This means ECMP is not suitable for multipathing across a wide-area network. Even if ECMP is relaxed and extended to work in WANs, it would still have to handle the problem of unequal bandwidth between multiple paths, which can either cause out-of-order packets or packet loss, none of which can be realistically handled by the network layer alone.

2.2.2 Multipathing in Application Layer

GridFTP [1, 2] is one example of a multipath protocol, in that it uses multiple TCP streams in parallel to improve performance along the network topology [16] by extending FTP with additional commands to support parallel streams. Application-layer multipathing can provide especially strong control over quality of service and policies regarding what would be higher in list of priorities to communicate first, such as prioritizing the more important parts of the data first, then sending the rest as appropriate. It is most suitable for complex applications that have a large variety of tasks, or those directly designed for multipath network environments such as high-performance computing, clusters, and so on.

While useful and easily controlled by applications, application layer multipathing can be error-prone [4] and hard to maintain [10]. The task of working with the multiple paths and flows will fall upon the application, which is not aware of the many mechanisms that are already available and working in the transport layer [25]. Some mechanisms that involve specific transport-layer packets (TCP *SYN* and *ACK* are common examples) may not work at all when implemented in applications, because they do not see the complete information necessary that support such functionality. Additionally, applications not written with multipathing in mind will not benefit when multiple paths are available. In the case of small applications or those with little network activity, implementing multipath networking may not feasible as the problems far outweigh the benefits.

2.2.3 Multipathing in Transport Layer

For many of the applications that would not be suitable for application-layer multipathing, it is more desirable that the paths be managed from the transport

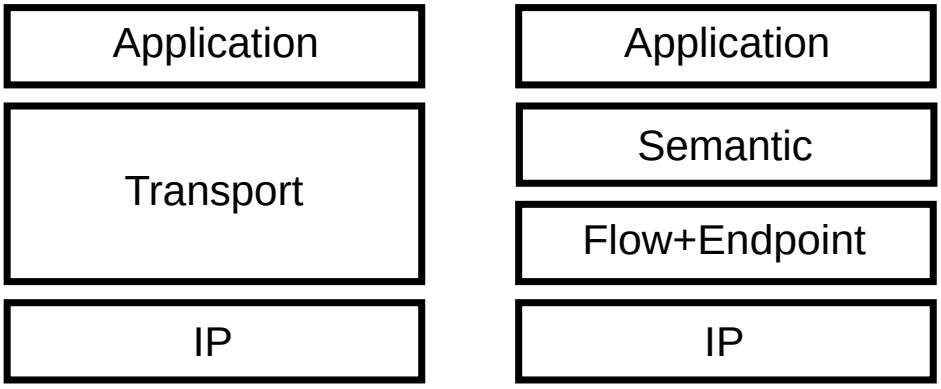
layer, which is more informed about each path than applications [4] but is also aware of the high-level connections not accounted for in the network layer. While SCTP, a transport-layer protocol, is also capable of multipathing, the feature is used only for redundancy purposes and not to increase bandwidth utilization [12]. Additionally, since SCTP is a completely different protocol, it may be treated differently by network devices, and applications still need to be modified in order to use it.

Even if actual multiple paths are not available, using multiple TCP streams in parallel over a single network path may still be desirable. The reason is tied to the congestion control mechanism of TCP itself that results in a constant cycle of high and low bandwidth. Multiple TCP streams sharing the same path will split the bandwidth and apply congestion control independently, resulting in reduced spread between high and low moments and therefore a more stable throughput. [17].

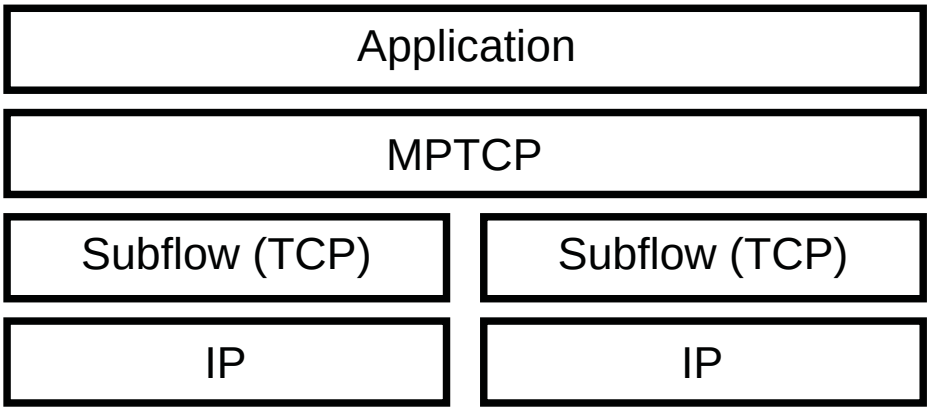
Many researchers, as a result, look at transport layer as a viable position to multipath. Many works such as *concurrent TCP (cTCP)* [10], *M/TCP* [37], and *Heterogeneous Multipath Transport Protocol (HMTP)* [23] provide multipathing solutions in transport layer. Many of these are mentioned in [7]. Among these, MPTCP [13] was considered an interesting protocol as it is implemented as TCP options and not a separate protocol, allows paths to be added and removed while the connection is still established, allows multipathing to be initiated from sender or recipient which allows it to work in a greater variety of environments, manages congestion control as a group (not per individual TCP connection), and has integral security features. It also has extensive research, including works on making a Linux kernel implementation [35] and a large following of community.

2.2.4 Multipath TCP

Multipath Transmission Control Protocol, or *MPTCP* is an extension to *TCP* at the transport layer to utilize multiple paths between two network endpoints by opening multiple *subflows* based on the number of interfaces in the host. Each subflow then behaves like a TCP flow, with its own congestion control, send and receive windows, and so on. Multipathing allows us to use more than one path in one logical connection, increasing bandwidth utilization, improving redundancy



(a) TCP/IP Model (b) Decomposed-TCP Model



(c) MPTCP interpretation of the decomposed layers.

Figure 2.2: Relationship between the traditional TCP/IP network model and derivation that led to the principle design of MPTCP.

and stability, as well as allowing seamless handovers in certain environments. It is especially useful in multi-homed networks and systems, which are more prevalent today.

MPTCP network model

By decomposing the transport layer into two sublayers, as shown from Fig. 2.2a to Fig. 2.2b [14], MPTCP can separately recognize the end-to-end and point-to-point situations better than the traditional model by having the upper half,

which is the MPTCP extension, work only with managing the connection and subflows, while the lower half works like ordinary TCP, dealing with congestion and other matters in each subflow as in Fig. 2.2c.

Advantages and Limitations of MPTCP

However, even with all the advantages MPTCP could offer, there still is no guarantee that the multiple paths employed (a *"path set"*) will always be the most optimal *path set* possible, as the paths in a *path set* may overlap or collide with each other reducing the leverage provided by MPTCP. Since MPTCP works on the transport layer and operates only from the source and destination hosts, it does not have control over the path selection at network layer. Therefore, an additional mechanism is required to guide MPTCP traffic onto multiple paths.

2.3 Multipath Routing

While OpenFlow and MPTCP each has its own limitations, combining them to work in a single implementation could make them work well with each other. The traffic engineering and routing techniques provided by OpenFlow can yield optimal *path sets* that MPTCP can use to manage multiple subflows while presenting a single interface to the application, allowing maximum performance to be achieved with minimal additional effort from the application. While such solutions have already been proved workable in [34], we have a further plan to optimize it specially for wide-area networking and distributed data storage environments, as well as to develop and improve the path selection strategy with regards to other factors beyond available bandwidth.

2.4 Distributed Storage

Distributed storage system (DSS) and distributed file system (DFS) refer to networked systems of multiple storage nodes or servers working together to provide storage capability to the users, which could be an actual physical user (e.g. a human being operating a computer) or another computer system that stores and retrives the files from DSS/DFS.

DSS/DFS can be seen as a foundation to construct a logical view of multiple file servers on the network [45] and provide a translation between the logical view and the physical storage structure of the files in the system.

Apart from the naming practice of their own developers, the distinction between the terms "distributed storage system" (DSS) and "distributed file system" (DFS) are not given much attention. In a specific report, DSS is considered to originate from or consist of DFS [33], and another report uses only the term DFS in a catch-all fashion [45]. Therefore, it is reasonable to regard DSSes as being a broader unit than, or implemented using, DFSes, but the actual terms may be used interchangeably depending on context. For the purposes of this disseration, the term DSS will be used for consistency.

For the purposes of this research, a number of well-known DSSes were reviewed. The works included in this section are Network File System (NFS) [42] *, Ceph [49], Lustre [5], GlusterFS [9], Hadoop Distributed File System (HDFS) [40], Google File System [15], and Gfarm File System [43]. The following subsections describe the general landscape of these DSSes and how they can be roughly classified based on their functions.

2.4.1 Architectures

There are many ways to provide a taxonomy on the architectures of DSSes, but among the easiet and clearest ways are to divide them between *client-server model* and *cluster-based model*, and also between *centralized* and *decentralized* architectures.

In addition to these classifications, the storage methods of the DSS, commonly trisected into *file-based*, *block-based*, and *object-based*, are also to be taken into consideration. *File-based* storages organize files and their metadata in a directories-and-files fashion, which makes the system simple to implement and easy to incorporate with various user policies such as access levels and permissions. However, they are usually seen as being inferior in performance compared to *block-based* systems because the latter manages data directly on a block-by-block basis, man-

*Newer versions of NFS includes additional features beyond the initial standard. For the purposes of comparison, only the core feature of the classic versions of NFS, which is the mounting and sharing of storage, are considered.

aging a minimal amount of identifier without considering any sort of metadata. *Block-based* systems are therefore superior in terms of flexibility, usefulness for applications that do not work with the logical structure of a file (e.g. databases). However, the complexity of implementation makes it impractical for cases where file-based storages are sufficient. A relatively new and emerging type of storage, *object-based* storage, treats metadata and data as objects that can be given flexible identifiers and stored in the system. These systems usually provide an application programming interface (API) that allows the data to be flexibly accessed by users or applications. This kind of storage provides a relatively high level of abstraction that makes it useful for scalability, as the management of metadata and file location can be handled by the (group of) servers responsible for them.

Architecture classifications of the DSSes are considered for the candidate of the DSS to be used in this research.

2.4.2 Bandwidth and Availability Requirements of Distributed Storage Systems

Primary points of concern for DSS performance includes bandwidth and availability, which are addressed by various mechanisms of the DSSes themselves or by providing an auxiliary mechanism or an extension to improve the usefulness.

DSSes are supposed to support a large number of users using a large amount of bandwidth. Exabyte-level system implementations are becoming more commonplace as big data and supercomputing expands [36]. As "exascale" computing is constantly evolving and the concept of supercomputing itself now relies more on DSS, it is clearly evident that the performance of DSS is in need of further research attention.

Many *centralized* DSSes suffer from the problem of single point of failure (SPF), which may vary depending on the DSS. For example, non-replicative systems such as NFS, the file server itself is the SPF. For systems with data replication but highly centralized metadata servers such as HDFS, the system could fail when the limited number of metadata servers are unavailable. On the other hand, systems that distribute both the files and metadata such as GlusterFS and Ceph

can sustain more system outages until the cluster is inoperable.

However, distribution of data and metadata has a price. As with any sort of distributed system, DSS must be able to maintain the consistency of the file copies, making sure that every data is updated, manage simultaneous write and read operations, synchronizing between the metadata servers if there is more than one, and so on. For simple centralized systems such as NFS, it is easy to manage file consistency because there is only one copy of a file, and the operating system is the only entity maintaining the file index and lock, so a separate metadata server is not necessary.

There are many reasons that a single storage node can fail. For example, the network could be disrupted as the switches or routers become overwhelmed or otherwise unavailable. Hard drives can go down and need to be replaced. Even with replicative RAID levels such as RAID 1, it is still important to take the node down for inspection and repairs when a hard drive fails. There is even an instance where a commercial storage provider treats hard drive failures as statistics and occasionally make them available to the general public [†].

The point of DSS resilience is not about keeping all the nodes up, but rather how to make the data still readable and writable when some of the nodes are down. Many DSSes are designed for data recovery, such as a well-placed replication of data across multiple nodes using various strategies, implementing restorative RAID levels (such as RAID 1) within each data storage server for additional stability, and having awareness of physical server settings such as the Rack Awareness concept implemented in HDFS.

During system and data recovery, it is important to complete the process as quickly as possible. Unless the entire DSS has a backup, it is not possible to lose access to all but one server and expect all data to survive. Many DSSes have a set number of replications per file, block, or object, and the loss of that exact number of server nodes can lead to permanent loss of the file, block, or object in question.

Furthermore, when an error occurs, correction happens in bursts. When a new hard drive is installed, data is quickly copied from other locations. When a large file is updated, a lot of synchronization occurs.

[†] `https://www.backblaze.com/blog/hard-drive-failure-rates-q1-2017/`

Depending on the distributed storage system and the storage method configured in the system, it is possible that one node's file may be replicated to multiple other nodes. When that node goes down and is restored, files from multiple nodes will be replicated back to the original node. While it is possible to reduce the number of servers involved in this process by designating two servers to be exact clones of each other, this could result in a very high load on the surviving server when another one goes down.

As we can see, there are many architectural considerations that must be made when considering a DSS for application. These unique behaviors would also lead to unique communication patterns between the nodes in each DSS.

2.4.3 Communication Patterns

Every distributed system has its unique communication pattern, and the exploitation of this information can lead to performance increase both by a meticulous network design that takes into account this pattern, as well as a routing technique designed to make communication within such system more efficient.

Some distributed systems already employ a two-network approach where one public and one private subnets are run separately. In this case, the public subnet is used to provide connectivity to the clients for receiving commands and sending files as well as status messages. On the other hand, the private network is used for synchronization of messages, file backup and recovery, as well as other housekeeping features in the system.

2.4.4 Rationale for Distributed Storage Choice in this Research

A variety of DSSes were reviewed in this work. A careful consideration was carried out between all of them to determine which would be the best fit as the primary target in the experiments.

While Lustre has an outstanding performance as a DSS, providing one metadata server (MDS) and storing data on separate object storage targets (OST) through object storage servers (OSS). Despite the high performance and flexibility, it was first ruled out due to the goals of this research. Since MPTCP was

planned to be used, and both Lustre and MPTCP require kernel modification to achieve full performance, it is a risk to validity and stability of the system to use two modifications at the same time in a single kernel. Due to the much higher rarity of viable and flexible multipathing systems compared to DSS, MPTCP was kept in favor and it was decided to find another DSS instead.

Network File System or NFS is among the most classic DSSes. It operates on a *client-server* model and basically it is a way to view and interact with a remote directory on a different server. In addition to this, since there is only one copy of the actual file which is stored in the central server providing the file itself, it can also be said that NFS is a *centralized* DSS. Since there is actually only one node running NFS at any given time, it becomes a single point of failure that can make the system unreliable. It is possible to counter this weakness by providing multiple servers sharing the same or similar set of files by using simple synchronization methods, creating a "hack job" kind of DSS with simple redundancy. However, this kind of crude management adds another layer of complexity because consistency, among other factors, suffers if the synchronization system is not present or not properly managed. Load balancing is also another matter which must be taken into consideration, as applications or users may not be aware of the multiple NFS servers, and decide to read from or write to only one server, causing a bottleneck in the DSS itself. NFS was briefly considered as a primary DSS target in this work, but due to the volume of research that has already been performed on it, as well as its *client-server, centralized* nature and lack of scalability, there is not much uniqueness and complexity to the communication pattern. Therefore, NFS is also considered not suitable for this research.

Hadoop's HDFS is a storage system for Hadoop. It is a *centralized* system that has up to two nodes used as NameNodes, and works with a scalable number of DataNodes, each of which storing a number of files, separated into blocks. The file name, tree, and mapping of blocks are all managed by the NameNodes. Due to the files being actually stored as blocks, the block-based performance can be very high, and the replication can make the system resilient. One important point of HDFS is that since the NameNodes must be consulted every time a new file is accessed, the performance of the block-based storage is still hindered by the file identification. It is noteworthy that this characteristic can be considered a minor

issue for systems with few large files as opposed to many small files. HDFS was also taken into consideration due to its impact and usefulness in e-science, high-performance computing, and data science. However, since this research aims to provide a solution that represents a more general DSS, HDFS was also considered to be too specific for its purpose.

Google File System (GFS, but abbreviated here as GooFS) was also at one point considered during the search for the representative target of DSS. However, GooFS is designed to work with especially large file for Google's big data endeavors. Given the importance of benchmarking the performance of DSSes on both large and small files, it was not considered for inclusion in this research at this stage.

GlusterFS (also GFS, but abbreviated here as GluFS) is one of the *decentralized* DSSes because it uses hashing to map the files, as opposed to using designated metadata system like HDFS. This eliminates the bottleneck problem and makes GluFS stand out as a high-performance DSS. Furthermore, it is a well-known and well-maintained DSS that has a large adoption and support (funded by Red Hat). However, the lack of a metadata system makes GluFS hard to represent the other DSSes that do have one, so it was also not considered as a target candidate in this work.

Due to the researcher's familiarity with Gfarm, along with a simple yet robust consistency and locking mechanisms, it was also considered for selection as the primary target in this research. However, its centralized nature with only one metadata server makes it not extensible to represent systems that may have more than one metadata server. Additionally, Gfarm is originally designed as a system to directly support parallel processing, not a pure DSS for the purpose of storage. Furthermore, the known problems associated with it as of writing this dissertation[‡] outweighed the qualities. Therefore, it is decided to not use Gfarm as a test target.

Ceph provides a flexible architecture that can provide object, block, and file storage in a single system. Ceph depends on Reliable Autonomous Distributed Object Storage (RADOS) [51] that provides a system that can contain multiple object storage devices (OSDs), each a full computer unit on its own. The flexible

[‡]`http://oss-tsukuba.org/gfarm/share/doc/gfarm/KNOWN_PROBLEMS.en`

concept of OSD allows RADOS to manage a heterogeneous DSS that has different system specifications on each OSD. An arbitrary, but recommended to be odd, number of monitoring nodes can be used to provide a map of the cluster and checks the state of the OSDs. Ceph/RADOS uses a concept of placement group (PG) that abstracts and simplifies the problems of file placement, allowing higher management performance in the system. Ceph uses CRUSH hashing algorithm [50] to provide a deterministic way to locate data instead of fully depending on the metadata server to provide all information to the client at all times. This reduces the load on the metadata servers and provides a higher metadata-level performance. Ceph can be summarized to be a *decentralized, cluster-based* DSS. Due to Ceph's flexibility, its low reliance on metadata access, relative popularity, friendliness to the operating system (as opposed to Lustre), active development (Ceph is also funded by Red Hat), and highly distributed nature that permits as many monitor nodes as needed, Ceph is chosen as the primary target of this research.

3 Preliminary Study of Multipath TCP on OpenFlow Network

The advent of Multipath TCP and OpenFlow has enabled many new network systems and practices to be deployed and utilized. Multipath TCP has been gaining an increasing traction as various applications and extensions are explored, including cellular and mobile networks [32], leveraging the API exposed to the applications for better performance [19], and extensions to improve user cooperation in LTE networks [52].

With the traditional network model, various topologies of network such as Fat-Tree (in DCNs) and mesh (in WANs) are provided or present, allowing multiple routes to be taken between a pair of nodes. However, as explained, the classical TCP/IP model allows only one path to be practically used at a time. MPTCP provides a way to efficiently provide multiple paths over the network in an efficient fasion and works well both in the DCNs and WANs, but no control over its routing, usually resulting in many paths colliding amongst themselves and limiting the actual performance. This section contains a preliminary study regarding a simple combination of MPTCP and OpenFlow.

3.1 Development Testbed

3.1.1 Design Goals

The goal of this experiment is to evaluate and find the effectiveness of running MPTCP on an OpenFlow testbed. OpenFlow would be used to for routing, while

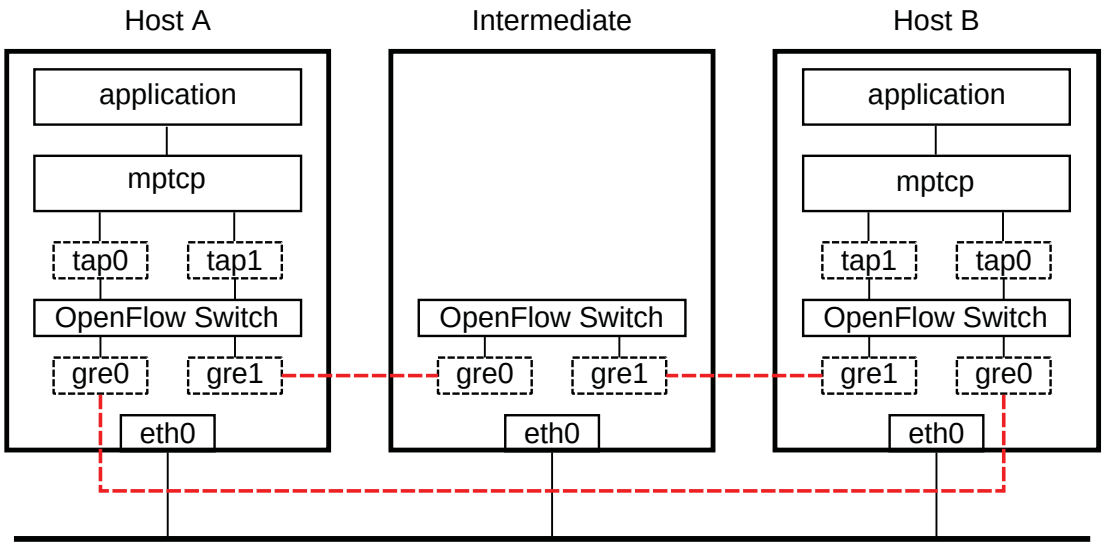


Figure 3.1: Implementation of the MPTCP/OpenFlow testbed.

MPTCP utilize them to distribute traffic across multiple paths to maximize the use of available bandwidth in the network.

3.1.2 Design Decisions

It is essential to maintain compatibility between the multipathing solution and applications. MPTCP preserves *vertical backwards compatibility* by providing the same interface to applications as TCP, so we think it is a viable method. If an application does not already use application-layer multipathing on its own, it will benefit from MPTCP. If the application already does, it can keep using its own multipathing mechanisms in conjunction with MPTCP. We expect the performance may be degraded, but the application should still function. It is also trivial to mention that MPTCP suite is compatible with IP, because TCP packets generated by TCP are already encapsulated in IP. Additionally, MPTCP is also *laterally backwards compatible*, able to revert back to TCP if a host does not have MPTCP installed. This is because MPTCP is implemented as a set of TCP options, initiated by sending `MP_CAPABLE` option during 3-way handshake. If a similar option is not returned, the initiating host immediately knows it has to fall back.

As mentioned in the previous section, MPTCP has no control over the paths

taken by each subflow, and a path set can be suboptimal. In order to use the most optimal multiple paths, multiple routes have to be created at the network level so MPTCP can utilize them. These paths are situated on different networks, and OpenFlow-based solution is used to provide the overlay network that fits this description. For this experiment, the flow entries were set up manually to provide static routing that works immediately without any sort of lookup algorithm and minimize the delay that would be caused by doing so.

3.1.3 The Testbed

The testbed was implemented as described in Fig. 3.1 on a VMware vSphere environment. Each node in this setup is deployed onto different host machines, interconnected with a total rate of 1 Gbps, shared between all users. The GRE connections established between each pair of hosts (red lines) are manually limited to 100 Mbps.

The MPTCP kernel (Linux 3.11) and related MPTCP utilities were obtained from the Multipath TCP Project [31]. The kernel provides MPTCP functionality while the other utilities provide configuration of MPTCP in various aspects, including preventing MPTCP traffic from spilling into unrelated network adapters. In this particular case, `eth0` interface was disabled so that MPTCP relies only on the GRE connections to communicate between the two hosts.

3.2 Evaluation Method

Network performance was evaluated by running ordinary testing mechanisms via `iperf`. This tool was chosen because only the measurement of raw throughput was desired. If MPTCP is active and working with unmodified applications, `iperf` should be able to utilize more bandwidth than without MPTCP. In addition to comparison against this common expectation, two variables were also created to determine their effects on the measured throughput. The first variable is the MPTCP State between on, on (but only one path is allowed), and off. The other variable is the number of `iperf` threads from 1 to 4. This yields 12 settings in total. Each setting is then run in five replicates, using 10 seconds for each repeat and 5-second pause periods between the replicates and the settings

Table 3.1: Bandwidth measured (in Mbps) between two host nodes over two GRE links. Each test condition is run for ten seconds for five times, five seconds apart.

	MPTCP, many paths	MPTCP, one path	w/o MPTCP
1 Thread	189.13	95.17	89.14
2 Threads	190.16	95.46	95.33
3 Threads	190.84	95.23	95.62
4 Threads	191.41	95.52	95.67

to make sure that the buffers do not interfere with the subsequent tests. The average measured throughput over time is then averaged for each setting. Transient bandwidth information is also obtained every 0.5 seconds.

MPTCP is integrated into the kernel and can be turned on and off using `sysctl -w net.mptcp.mptcp_enabled={1 or 0}`, while number of connections on `iperf` can be set by using `-P #`, where # is the number of parallel threads to run.

3.3 Results of Evaluation

Since the testbed is implemented on a virtual environment, it can be easily configured and modified. In this state, the network topology was kept as simple as possible, by having only two nodes with one intermediate. After running the experiment, data rates based on different conditions were captured as detailed in Table 3.1.

3.3.1 Bandwidth utilization of MPTCP

Based on the data rates shown in Table 3.1, it was discovered that by using MPTCP over two paths, the maximum bandwidth between the two hosts increased approximately one-fold. This means MPTCP can fully utilize the new path assigned to the two hosts. As shown in the second column in Table 3.1, MPTCP can function as well as without MPTCP even if there is only one path, suggesting that performance drop, if any, is negligible when using MPTCP. Addi-

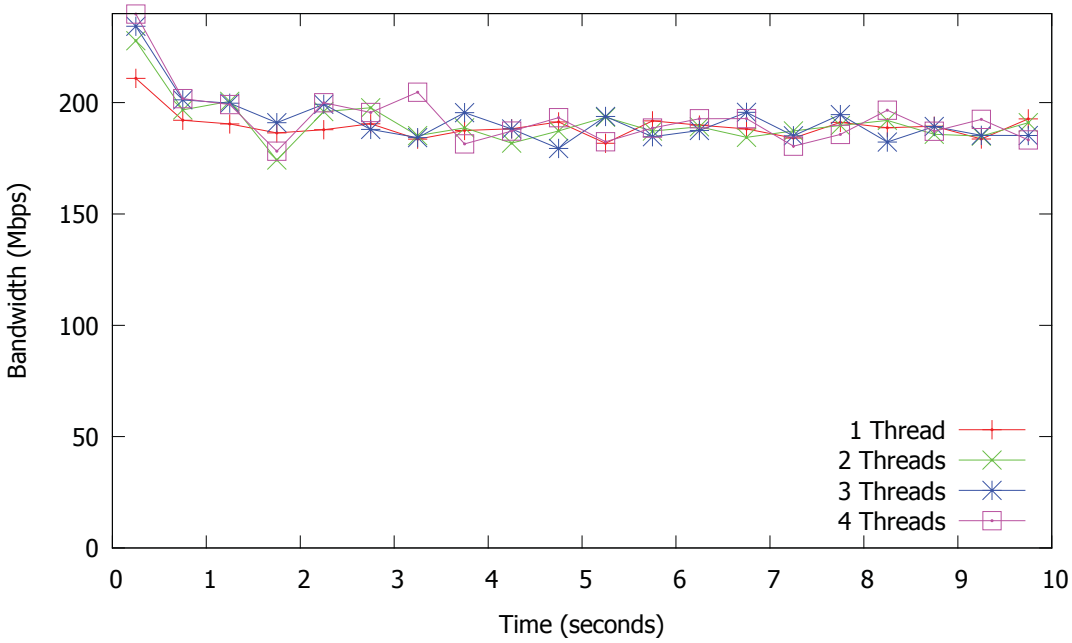


Figure 3.2: Transient throughput measured during 1-thread to 4-thread MPTCP tests.

tionally, MPTCP seems to be able to adapt to changing network conditions on its own. This should allow MPTCP to adapt to any kind of network if appropriate paths are provided by the means of OS- or network-level configuration.

3.3.2 Performance of a Single Thread

Based on the results in Table 3.1, it is worth mentioning that while the performance of running multiple threads with single-path MPTCP and without MPTCP at all yields the similar average values, the difference in the case of 1-thread evaluation was more significant (95.17 Mbps vs 89.14 Mbps). Since it was a difference of significance, a more thorough analysis was necessary. A transient plot of these two cases were made as shown in Fig. 3.3. The shape of the graph indicated that the case of running MPTCP in one thread showed a more stable throughput towards the maximum, compared to using only ordinary TCP without multipathing. This is the effect of TCP congestion control behavior which consists of starting slow, then increasing the speed of data transmission as

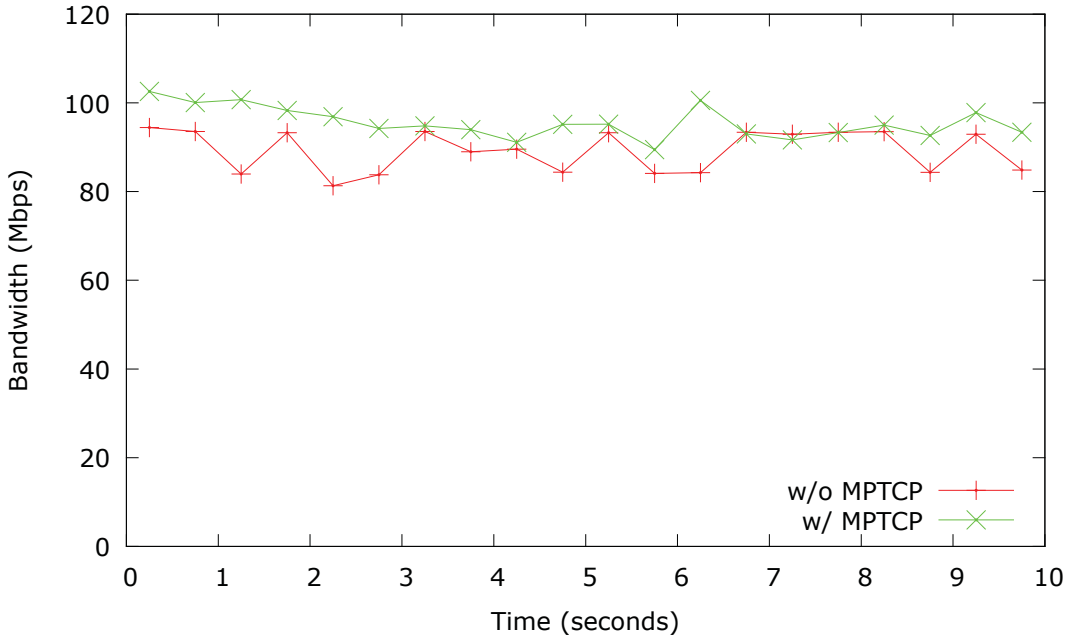


Figure 3.3: Transient throughput comparing the cases of running 1-thread test with and without MPTCP

the system works without packet loss [3]. The back-and-forth behavior of TCP congestion control itself could be the reason behind this result. It is possible that this difference would be even more pronounced when using a network with larger latency. All other settings, on the other hand, results in all the other cases with either one path or without MPTCP consist of subflows or flows sharing the single link, causing the aggregate throughput to reach the maximum more quickly.

In the case of running multiple threads without MPTCP, our results agree with a related work [17] which performed a similar kind of test. This work further suggests that in case of running MPTCP on a single path, a similar result could also be achieved.

Since this behavior is dependent on the congestion control, it is possible that mechanisms that do not exhibit this back-and-forth behavior or rely on packet losses, such as BBR [6], would not yield this kind of graph shape.

Table 3.2: Throughput measured (in Mbps) between two host nodes over two GRE links when one link is limited to 100 Mbps and another link to 50 Mbps.

	Without MPTCP	With MPTCP
1 Thread	47.82	142.37
2 Threads	47.60	143.94
3 Threads	47.86	144.54
4 Threads	47.86	146.52

3.3.3 Performance of Multiple Threads

In addition to the average values, the transient throughput was also measured by iperf every 0.5 seconds, as shown in Fig. 3.2. These results indicated that MPTCP can utilize bandwidth in a stable manner over both time and number of threads, no worse than ordinary TCP.

3.3.4 Performance in Unequal-Bandwidth Environments

Since it is normal for WAN links to be unequal in terms of bandwidth, we have also simulated this fact in our testbed by limiting one path's bandwidth to 50 Mbps as shown in Figure 3.5. The results shown in Table 3.2 indicate that MPTCP can fully function in unequal environment and can use multiple paths effectively. Without MPTCP, the performance would be affected if the single path uses the affected link.

3.3.5 Performance when Subjected to Disconnection and Reconnection

We also tested MPTCP's resilience by running a 60-second `iperf` session, which is called from Host A to Host B, but we shut down one of Host B's interfaces at 15 seconds, then another one at 25 seconds. We restored the original link at 28 seconds, then another one at 38 seconds. The experiment is run for 60 seconds in total and data points are collected every 0.5 seconds. The results, shown in Figure 3.6, demonstrates that MPTCP can respond to changes in path

availability. If one path goes down or becomes unavailable, another can be used, and additional paths can be used when they become available. This test was run five times.

3.3.6 Packet analysis

In order to check if MPTCP initialized and operated properly, we have captured the packets exchanged between the two hosts using `tcpdump` and put them into Wireshark, version 1.8.6. * Fig. 3.4 shows the first MPTCP packet with `MP_CAPABLE` option being sent from `iperf` client to server. By inspecting multiple packets in capture files, we also found out that multiple endpoints are used during communication. When each pair of endpoints is being initialized, an `MP_JOIN` option is sent in addition to normal TCP handshake.

3.4 Discussion

In this section, a group of MPTCP-enabled hosts implemented on an OpenFlow network was evaluated for the network performance. The hosts could utilize MPTCP and the network could use the primitive flow entries to increase aggregate application-layer bandwidth between two end hosts. It is also discovered that in this case of a virtual local network, MPTCP can work with no significant performance drop compared to ordinary TCP. Furthermore, even in the case of a single network route, MPTCP could serve to saturate the route more and provide a little extra throughput to the application.

By successfully designing and implementing a working MPTCP–OpenFlow combined testbed that yields superior bandwidth utilization and point-to-point data rate, we further reinforce the existing proof that MPTCP is viable as a drop-in improvement to TCP. This brings us closer to our research goal.

This, however, is still far from our main research theme. While `iperf` can simulate large file transfer, it still works on a host-to-host basis and still does not fully reflect distributed storage or other HPC systems.

*Wireshark supports MPTCP since version 1.7.1.

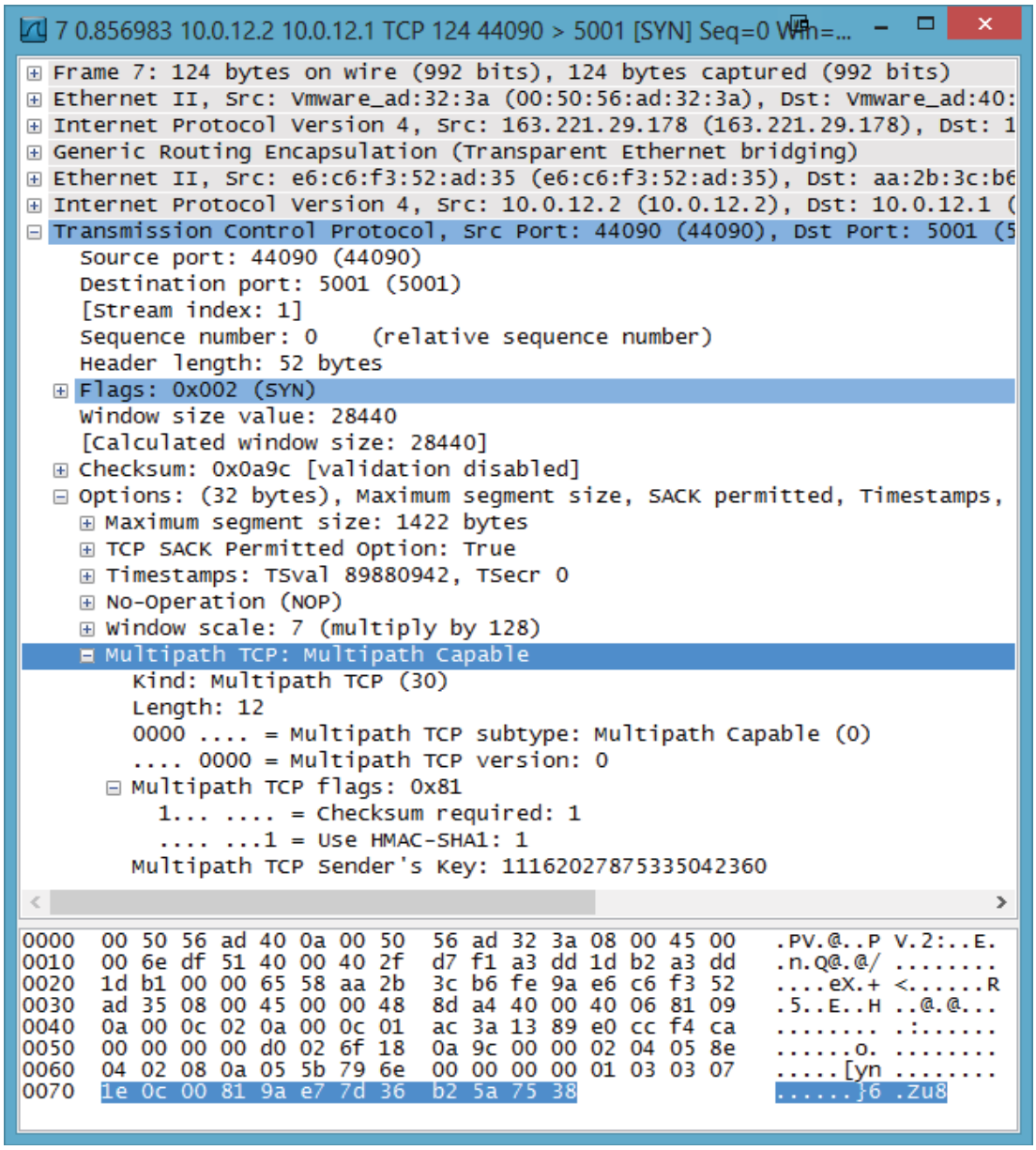


Figure 3.4: Wireshark analysis of the first packet captured while MPTCP connection is being established. Here, an MP_CAPABLE option is seen. GRE encapsulation between public and private IP addresses are also visible. The sender's key (last attribute shown in the top panel) can later be used as the basis for MPTCP flow group identification.

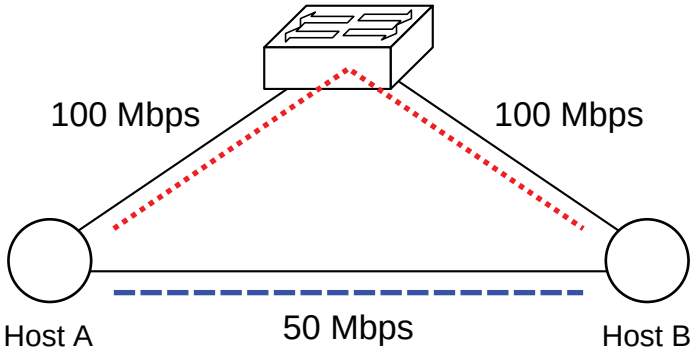


Figure 3.5: Composition of the hosts, the switch, and links between them. All links are limited to 100 Mbps.

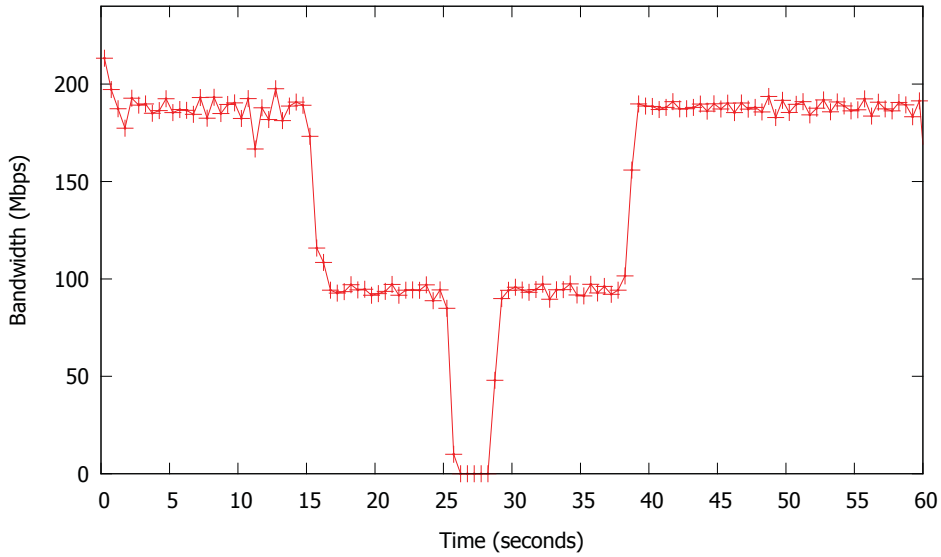


Figure 3.6: Transient bandwidth measured during the resilience test. MPTCP can fully utilize whatever resources that it can.

4 The simple multipath OpenFlow controller (smoc)

In this research, the simple multipath OpenFlow controller (smoc) was designed and implemented. Its early goal was to provide an OpenFlow controller that identifies the flow-subflow grouping in MPTCP, which is not possible from the perspective of packet routing alone.

4.1 Design of MPTCP Routing Mechanism

Two actions are necessary to route MPTCP traffic through the network using multiple paths. First, we need to know which subflows belong to which instance of MPTCP. Second, we also need to decide which paths would be used and when. These actions are further discussed in the following two subsections.

4.1.1 Identifying MPTCP subflow group on the network layer

As stated above, we need to identify which subflows belong to the same MPTCP instance. Since all MPTCP information is encoded as TCP options, not headers, it is impossible to just match the basic protocol headers to identify MPTCP subflow grouping. Therefore, a method to identify the subflows from the OpenFlow controller's perspective is necessary. In order to do so, special information beyond IP addresses and TCP port numbers is required.

Fortunately, MPTCP exchanges all need information during the initial MPTCP subflow establishment (using `MP_CAPABLE` TCP option) and subsequent subflows (using `MP_JOIN` option). This process is basically an extended version of TCP

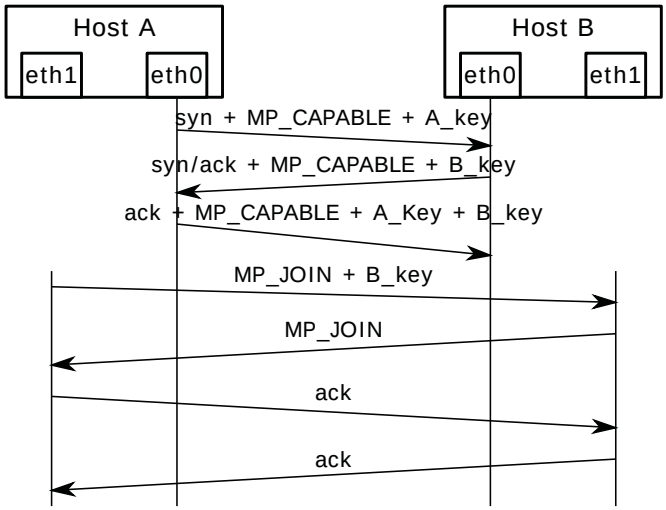


Figure 4.1: MPTCP handshake process, annotated with a partial list of TCP and MPTCP option fields used in our work.

three-way handshake, with MPTCP-related information added into the process. While this is, from a strict viewpoint, considered to be a purely L4 (transport-layer) exchange, it could also be considered as an "L4.5" communication. This is due to MPTCP information spanning through multiple TCP streams, forming a loose sort of session. From a hierarchy viewpoint, this distinction also makes sense because normally all TCP-related packets should belong to a specific connection. However, in this case, multiple TCP streams can belong to a larger group that is still not an application-layer (an "L5") protocol. From the monitoring viewpoint, it is understandable because tracking such "multi-connection" protocol would require following more than one TCP stream at a time, which takes significantly more effort than doing so on a single TCP stream.

MPTCP relies on *keys* and *tokens* to identify a connection endpoint which is unique for each connection and host. We can use this identification information to find which subflows belong to which MPTCP connection*. When MPTCP creates a new instance for the first time, each host sends its own key to the other host. When a host establishes an additional subflow, it (A) will send the other

*In MPTCP, keys are later hashed, truncated, and called *tokens*. As we do not work on the full process of MPTCP, we will not care about the differences between these terms. *Key* will be used throughout this paper for simplicity.

party (B)'s key to identify an MPTCP session it (A) wishes to join. As this process uses different IP address and TCP port pairs, an OpenFlow `packet_in` message will be sent from an OpenFlow switch to the controller, which would use this information. This process is illustrated in Figure 4.1 and is used as a basis for flow detection and grouping in our routing algorithm.

4.1.2 Finding and using multiple paths

Apart from correctly identifying MPTCP subflows, the paths are also required to correctly guide the flows through the optimal routes. In this work, the OpenFlow controller would use the topology information to find optimal *path sets*, a collection of paths that lead a packet from one host to another, and decide which path an MPTCP subflow should use. This mechanism involves multiple stages: first we analyze the packet and gather or match information with the database. In this step, a new path set may need to be created. Then, the path set would be applied to new MPTCP subflows as they are created. By cycling through the different paths in a path set, MPTCP subflows can be distributed to multiple paths.

Path Set Calculation Algorithm

Algorithm 1 Algorithm to find a path set to route from S1 to S2 in network graph G

Require: graph $G(V, E)$
Require: $S1$, $S2 \in switches$
  $G(V, E) \leftarrow NetworkTopology(switches, links)$
  $PrimaryPath \leftarrow$ shortest_path(G, $S1$, $S2$)
  $AltPaths \leftarrow$ all_simple_paths(G, $S1$, $S2$) - $PrimaryPath$
  $AltPaths \leftarrow AltPaths$ sorted by number of edges shared with PrimaryPath ascending, then by length of path ascending
  $PathSet \leftarrow PrimaryPath + AltPaths$
  return $PathSet$

Algorithm 2 Details of database schema. ** (two asterisks) denotes data types that may be dependent on actual implementation. PK denotes the attributes that constitute the primary key.

```
pending_capable = (
tuple(init_ip IPADDR**, init_port INT) PK,
tuple(listen_ip IPADDR**, listen_port INT) PK,
init_hash INT,
pathset COLLECTION-OF-LISTS**
)

pending_join = (
tuple(init_ip IPADDR**, init_port INT) PK,
tuple(listen_ip IPADDR**, listen_port INT) PK,
listen_hash INT,
pathset COLLECTION-OF-LISTS**
)

mptcp_connections = (
to_hash INT PK,
from_hash INT,
pathset COLLECTION-OF-LISTS**
)
```

Algorithm 3 Flow group identification algorithm to handle incoming MPTCP packets that trigger OpenFlow packet-in message.

Require: *packet*
Ensure: *route* for the flow of *packet*
pending_capable, pending_join, mptcp_connections defined in Algorithm 2
if *packet* is `MP_CAPABLE` message **then**
  if (*packet.dst_ip_port*, *packet.src_ip_port*) in *pending_capable* **then**
    recvkey, *ABpathset* ←
      pending_capable[(*packet.src_ip_port*, *packet.dst_ip_port*)]
    BApathset ← find new pathset
    add *recvkey* → (*packet.sendkey*, *ABpathset*) to *mptcp_connections*
    add *packet.sendkey* → (*recvkey*, *BApathset*) to *mptcp_connections*
    delete (*packet.dst_ip_port*, *packet.src_ip_port*) from *pending_capable*
    return *BApathset*.next()
  else
    ABpathset ← find new pathset
    add (*packet.src_ip_port*, *packet.dst_ip_port*) → (*packet.sendkey*, *ABpathset*) to *pending_capable*
    return *ABpathset*.next()
  end if
else if *packet* is `MP_JOIN` message **then**
  if (*packet.dst_ip_port*, *packet.src_ip_port*) in *pending_join* **then**
    sendkey, *ABpathset* ← *pending_join*[(*packet.src_ip_port*, *packet.dst_ip_port*)]
    recvkey, *BApathset* ← *mptcp_connections*[*sendkey*]
    delete (*packet.dst_ip_port*, *packet.src_ip_port*) from *pending_join*
    return *BApathset*.next()
  else
    sendkey, *ABpathset* ← *mptcp_connections*[*packet.recvkey*]
    add (*packet.src_ip_port*, *packet.dst_ip_port*) → (*sendkey*, *ABpathset*) to *pending_join*
    return *ABpathset*.next()
  end if
end if
return shortest path as *route*

Algorithm 1 describes a simple method to find a path set for multipath use. When supplied with a network topology graph and the source and destination switches, the algorithm chooses one shortest path as the *primary path*. The remaining paths are sorted and prioritized to minimize path sharing with the primary path, and then by path length. We use the shortest path and all simple path functions from `networkx`[18] package.

Representation of Path and Path Set Information

Each path is represented by a simple list of all DPIDs, stored in integer, from start to end, including the first and the last DPIDs. Path sets consist of one or more paths stored together as a larger collection. For the implementation, the path sets are stored as Python `itertools.cycle` objects, which allows us to easily cycle through all paths inside.

Since only the specification of data is given in this section, it is also possible to replace the routing algorithm presented in Algorithm 1 with others, allowing the controller to be customized or improved with minimal changes to existing code.

Collecting and Managing MPTCP Subflows

To track the states of MPTCP subflows, three tables, with their schema represented in Algorithm 2, are used to store and match the subflows and assign them to routes by Algorithm 3.

The first table is the `pending_capable` table which stores information of first SYN packets sent by the MPTCP initiator using `MP_CAPABLE` message. It maps the IP address and TCP port to the initiator's hash and also stores the path set from the initiator to the listener.

The second table is the `pending_join` table which does a similar function for subsequent subflows created by `MP_JOIN` messages.

The final table, `mptcp_connections` table, stores a list of already established MPTCP connections. Once an entry in the previous two tables is matched by a reply packet (TCP ACK), that entry is removed from its original table and the path set will be stored here. It maps a destination's key to the source's key and the path set from source to destination. Since, by the style of OpenFlow connections, it is not possible to detect when a connection is over, the current

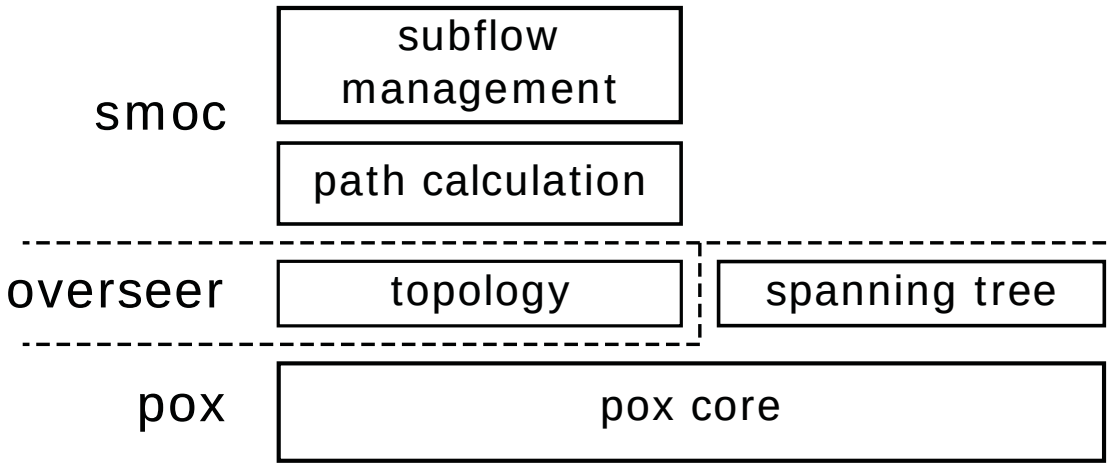


Figure 4.2: Components of the smoc controller, based on POX framework and Overseer's topology management modules.

implementation smoc is configured to hold a maximum number of connections that expires over time. New `MP_JOIN` subflows are matched against this table. This method makes it possible for a monitor or observing entity to tap into the information of MPTCP handshakes, which remain entirely end-to-end and initiated by the hosts, without affecting their operations or acting as a proxy.

4.2 Implementation of smoc: Simple Multipath OpenFlow Controller

To achieve the goal of solving the multipath bottleneck problem by using OpenFlow to program the flow tables for MPTCP, we implement the algorithms described in Subsection 4.1.2 in our controller, Simple Multipath OpenFlow Controller (smoc). The core of smoc is based on POX, a well-known OpenFlow controller framework. POX was chosen due to its modularity which means new features can be rapidly developed. Topology management and path management features are based on Overseer [48, 47] which is also an OpenFlow controller based on POX. Overseer's original purpose is to optimize routing based on characteristics of applications. To serve its purpose, Overseer has well-designed topology management and path management features, which also form the basis for smoc.

Path finding is assisted by the `networkx` Python package while path selection is based on Algorithm 1. smoc implements Algorithm 2 as expiring dictionaries (ExpiringDict[†]) in the now-modified running instance of Overseer. The majority of the `_handle_openflow_PacketIn` method was replaced by an implementation of Algorithm 3 to provide a multipath-aware subflow management system instead. Underlying maintenance functions such as spanning tree management and communication with the switches using the OpenFlow protocol are handled by POX and Overseer.

To route flows, we maintain a list of pending and connected sessions. When we receive a new connection handshake message, we calculate a new path set and add both the information of the connection initiation and the path set to the pending list. When the pending connection is responded, we calculate another path set for the reverse direction and move everything to the connected sessions list. Any subsequent connections would only require a lookup in the connected sessions list to find an appropriate path.

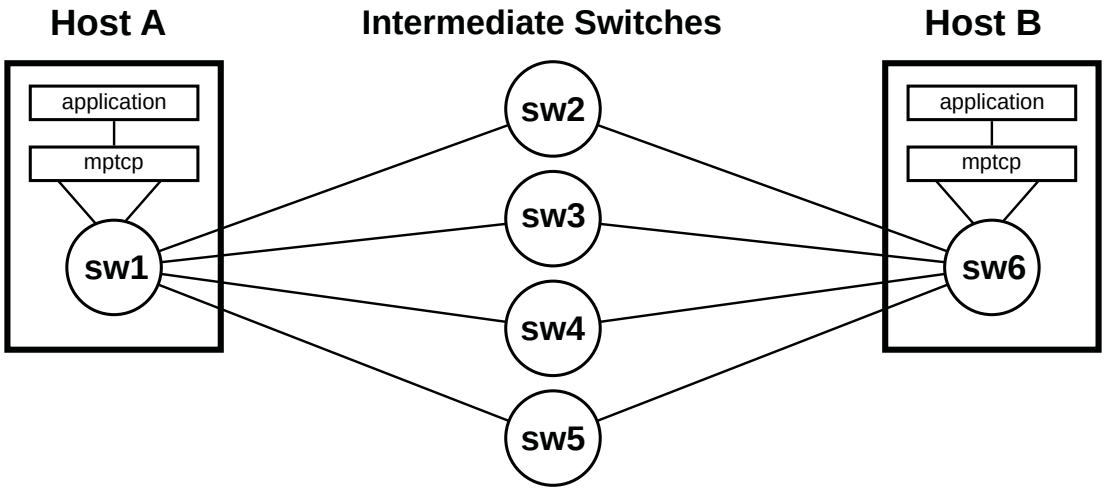
4.3 Evaluation and Results

We evaluated smoc against POX's original spanning tree controller (henceforth, POX S-T). With this controller, all MPTCP traffic would be confined to a single path even if multiple paths actually exist in the network. We chose this controller because it is based on the same framework and architecture, and spanning tree is commonly used to prevent loops in network topology. However, spanning tree eliminates any sort of multiple paths that exist at the network topology level. This means POX S-T always produces a single path between any pair of hosts. Being based on the same technology as smoc, all basic program libraries would be the same. This makes POX S-T suitable for an experimental control.

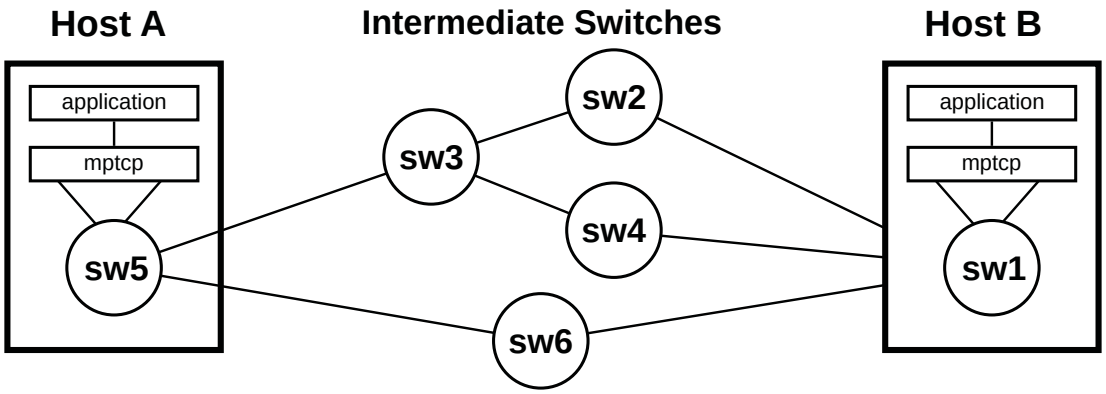
We chose `iperf` as our benchmarking tool due to its simplicity. smoc was evaluated in two testbeds, a local-area and a wide-area testbed. These testbeds, based on our previous work in [28], represented different network environments.

In the local testbed, two topology configurations shown in (Figure 4.3) are modeled after a previous work from our research group [21]. Topology 1 (Fig-

[†]`https://github.com/mailgun/expiringdict`



(a) Topology 1 on the local testbed



(b) Topology 2 on the local testbed

Figure 4.3: Topology configuration of the local testbed. The switches are Open vSwitch installed on virtual machines. Each virtual machine is hosted on a separate physical host. Links between the switches are limited to 100 Mbps.

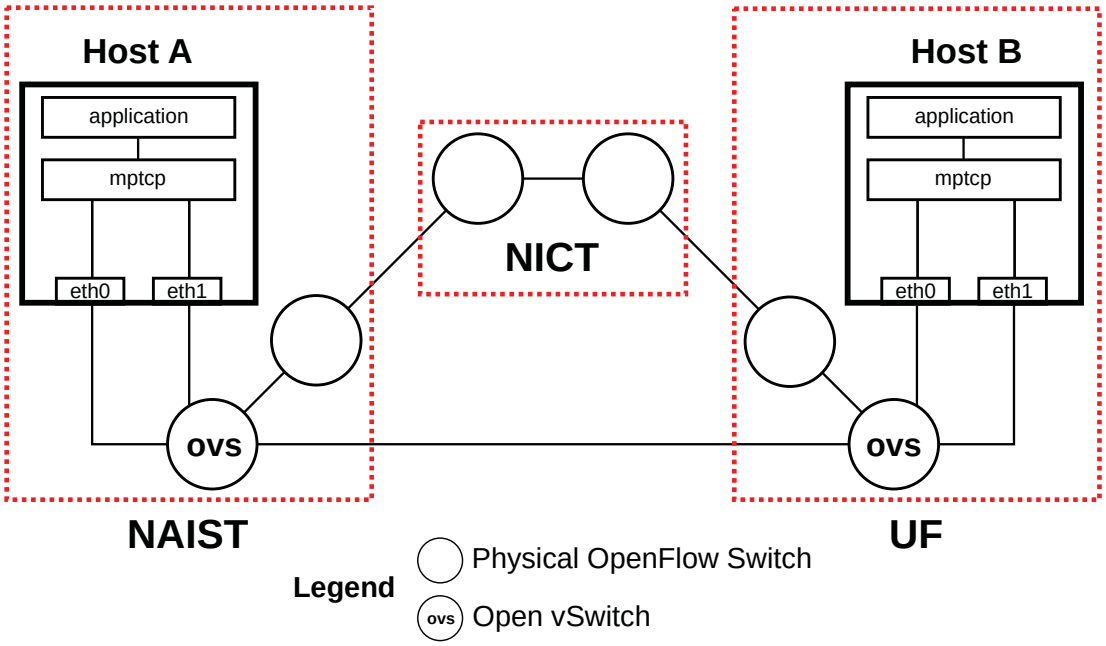


Figure 4.4: Testbed implementation in PRAGMA-ENT. The hosts are installed as virtual machines on the two sites.

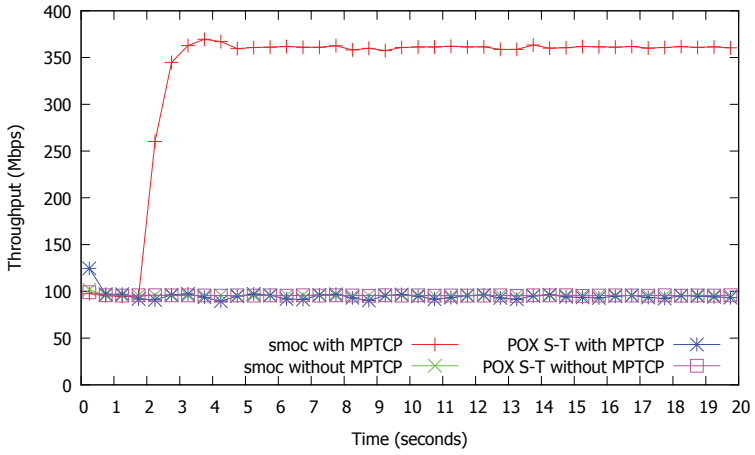
ure 4.3a) has four isolated paths, while Topology 2 (Figure 4.3b) has paths partly sharing a link.

The wide-area testbed (Figure 4.4) experiment uses an existing collaborative wide-area software-defined network project known as the Pacific Rim Applications and Grid Middleware Assembly Experimental Network Testbed (PRAGMA-ENT) [38].

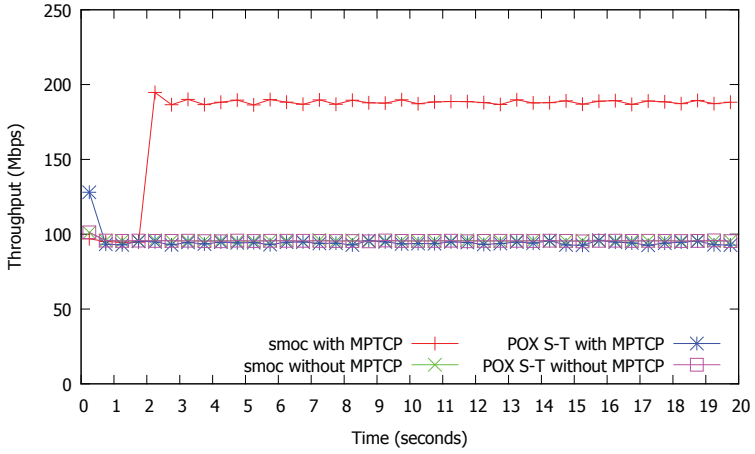
4.3.1 Evaluation in virtual local-area SDN

We implemented our local-area testbed on a VMware vSphere environment using six virtual machines. Each virtual machine, containing MPTCP installation and Open vSwitch [30], were deployed to different physical host machines. The GRE connections established between each virtual machine are manually limited to 100 Mbps to ensure that our virtual environment has a stable and clear maximum level of bandwidth, allowing easier verification of MPTCP and our controller.

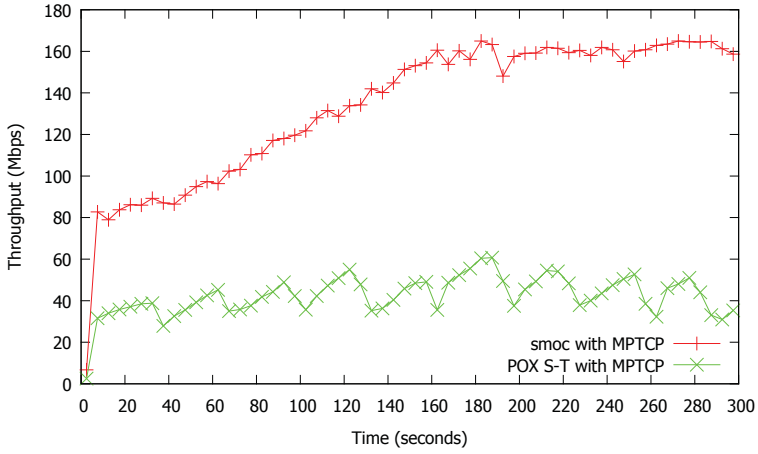
We obtained MPTCP kernel and utilities from [31]. The kernel provides



(a) Results of test on the local testbed with Topology 1



(b) Results of test on the local testbed with Topology 2



(c) Results of test on the wide-area testbed with PRAGMA-ENT

Figure 4.5: Transient throughput between two hosts measured by `iperf` on different network topologies

MPTCP functionality while the other MPTCP utilities allow us to disable MPTCP on select interfaces to make sure that the experimental traffic does not "spill" into the management subnet. This MPTCP kernel comes with multiple options that can be set through the `sysctl` variables, allowing us to customize the subflow creation options and numbers. Some options allow an arbitrary number of subflows to be created, regardless of the actual number of interfaces of the machine.

Testing POX S-T and smoc produced results as shown in Figure 4.5. Without a combination of a multipath router and MPTCP, only one path could be used at a time and the test run showed only approximately 95 Mbps of bandwidth, slightly below the 100 Mbps limit, was used. However, when smoc and MPTCP are used together, after a few seconds of delay in the controller, the measured bandwidth was increased to greater than 100 Mbps, indicating that multipathing was successful with this combination. Test results using Topology 1, shown in Figure 4.5a, indicate that all four paths between Host A and Host B were used, allowing the maximum throughput to reach up to 400 Mbps. Test results using Topology 2, shown in Figure 4.5b, the measured throughput reached the maximum aggregate bandwidth of 200 Mbps as configured.

4.3.2 Evaluation in physical wide-area SDN

Two virtual machines were used for the evaluation in the wide-area SDN. One was deployed in NAIST (Nara Institute of Science and Technology), Japan. Another was deployed in UF (University of Florida). Two paths were used in this experiment. For the first path, NAIST and UF are connected through two physical OpenFlow switches provided by NICT (National Institute of Information and Communications Technology), Japan. For the second path, a GRE link was directly established over the Internet between NAIST and UF.

smoc outperformed POX S-T from the start, then continued to increase its throughput throughout the test as shown in Figure 4.5c. It is noteworthy that since TCP increases window size slowly in wide-area networks due to long round-trip time, more experiment time is needed for smoc to reach the maximum bandwidth possible in the network. For the wide-area test in PRAGMA-ENT, we used 12 `iperf` threads (-P 12) to saturate the network, using more bandwidth. This means `iperf` produces more consistent values toward the maximum available

bandwidth.

4.4 Discussion

In this section, we discuss the performance of our algorithm, issues with path installation delay in our controller, conditions of the test environment, and scalability of our solution.

4.4.1 Algorithm performance

We used a purely topological routing algorithm and generated path sets based on "minimum shared edges – minimum hops" basis. Whlie this is very simple to implement, only requiring a few calculations and no monitoring at all, the performance in real-world WANs may be debatable as topology information alone is not enough to effectively route flows through the best paths. One quick improvement that could be done to this controller is to use bandwidth-based routing by implementing a weighted graph and bandwidth monitoring to supply the graph with weights. Passive bandwidth monitoring was considered because we do not require the level of precision that could only be achieved by active monitoring. Any changes to the topology in real-time would be noticed by the management modules provided by POX and Overseer.

4.4.2 Path installation delay

We experienced a 2-3 second delay in path installation as seen in Figure 4.5. This delay is caused by the path installation process by underlying POX modules. While this delay may be insignificant when a flow is long enough, it may impact short flows and cause scalability problems when handling a large number of flows. We need to find some way to improve the performance of the controller, such as shifting from the current reactive approach to a more proactive one which is more scalable [11] and has better performance. Some examples of proactive measures possible for smoc include anticipating and preinstalling flow entries for additional subflows right after the first subflow is created, or storing a group of frequently-used path sets so they do not have to be calculated every time a new flow enters

the network.

4.4.3 Test environment

While PRAGMA-ENT is a very good representation of WANs, the segment that we used consists of only two paths and a small number of switches. Even if these switches represented many more actual network elements, a more complex network could prove beneficial to the evaluation of our work. Additionally, testing with real-world applications would provide a realistic picture of our experiment. The high latency present in PRAGMA-ENT caused TCP flows to increase their window sizes more slowly. Shown in Figure 4.5c, it takes about 160 seconds for smoc's TCP flows to collectively increase their throughput to about 160 Mbps. This means spending more time with the test runs on high-latency networks should provide clearer results.

4.4.4 Scalability

Even though the multipath routing algorithm described in this work is adequate to efficiently route a set of subflows belonging to an MPTCP session through multiple paths, the smoc controller itself may have scalability problems. smoc is inherently centralized due to its use of OpenFlow. It has been studied that the number of flows that can be processed by the OpenFlow controller reduces at a quadratic rate with increasing number of switches, regardless of using proactive or reactive approach in routing [11]. As described earlier, reducing path installation delay by using proactive routing and reducing path set computation time can be some simple ways to mitigate (but not completely eliminate) the scalability problem by increasing the rate of flow processing. More involved methods include considering additional features in later OpenFlow protocol versions, such as TCP flag matching introduced in version 1.5.0, to allow the switches to make more decisions on their own without requesting decisions from the controller. However, not all switches support the newest versions. We must consider the compatibility between the controller and the target environment carefully before upgrading the protocol version used in our controller.

Apart from the methods mentioned above, we could consider alternatives and

modifications to OpenFlow, such as HyperFlow [46] and DevoFlow [8]. HyperFlow uses multiple synchronized OpenFlow controllers to communicate with each other and split the workload. Installing one HyperFlow controller per site may lead to greater scalability than using a single OpenFlow controller for the entire network. On the other hand, DevoFlow, which is a significant modification to OpenFlow, aims to reduce workload on the controller by allowing additional actions on the switches, such as rule cloning and multipath support. These solutions would be able to improve scalability of many existing OpenFlow applications including smoc.

4.4.5 Adjustments to Subflow Assignment

It is possible to configure MPTCP and the network to create more subflows than the actual number of possible paths. In this case, at least one path will be used by more than one subflow. By considering network characteristics such as latency and bandwidth and configure our controller accordingly, we might be able to increase total throughput. For example, placing more subflows on a path with larger bandwidth would allow those subflows to take advantage of the bandwidth, rather than having them contend against each other on a smaller bandwidth path.

4.5 Conclusion

In this work we presented a simple multipath OpenFlow controller that routes MPTCP connections by splitting them across multiple paths. Tests on both LAN and WAN SDN testbeds yielded positive results, indicating that our controller works as intended. No modifications to applications or host machines were made (only the kernel in the virtual machines), making our solution backwards-compatible with existing systems. We would find ways to improve its performance in future iterations of our work.

5 Analysis of Distributed Storage Communication Patterns

5.1 General Setup of the Testbed

5.1.1 Equipment

The six server nodes were implemented as virtual machines in the six servers at Nara Institute of Science and Technology's VIS Center, the successor to ISA.

5.1.2 Software Versions

Each of the six experiment nodes are configured as follows:

- Ceph 10.2.9
- CentOS 7.4
- MPTCP 0.91.3 based on Linux 4.1.39
- Open vSwitch 2.5.1
- OpenFlow 1.0
- Python 2.7.5

Ceph was chosen based on reasons in Sect. 2.4.4. CentOS was chosen over Ubuntu in this case due to software compatibility and similarity to the systems at PRAGMA, which are based on Rocks Cluster and belong to the Red Hat family of Linux distributions. MPTCP was updated specifically to this version with bleeding-edge new kernel due to the advanced congestion control features and

additional settings not available in the previous versions. Open vSwitch 2.5 version is the most stable and newest at the time of writing this dissertation (newer versions were attempted, but they produced unstable results). Open vSwitch itself was chosen over using physical switches due to the virtualization of the testbed itself. OpenFlow was strictly limited to 1.0 to guarantee compatibility with OpenFlow-capable hardware, which may or may not support newer versions like 1.3. At the time of writing, Python 2.7.5 was the latest version officially available (through official repositories) to the distribution.

The OpenFlow controller uses the following software:

- Ubuntu 14.04.5 LTS
- Python 2.7.6
- POX 0.5.0
- NetworkX 2.1

The controller retains Ubuntu 14.04.5 LTS as it was configured a long time before the experiment nodes themselves (which were renewed for new versions of CentOS and MPTCP in mid-2015). Python and POX were the newest officially available versions on this distribution. NetworkX 2.1 was chosen and is strictly required (i.e. 2.0 will not work) due to the addition of some advanced path calculation methods. NetworkX itself was chosen over other network or graph-theory libraries due to its high performance, functional completeness, general widespread use, and reliability of software support.

5.2 Evaluation of smoc in a Ceph installation

The six nodes were configured as shown in Fig. 5.1. While this particular configuration is not optimal for ceph, it represented many other DSSes well because there are also many DSSes that employ a single metadata server and multiple OSDs. The two clients were situated as shown in the figure so that they can perform upload/download operations with one or two OSDs, and two clients can access a single shared OSD as well.

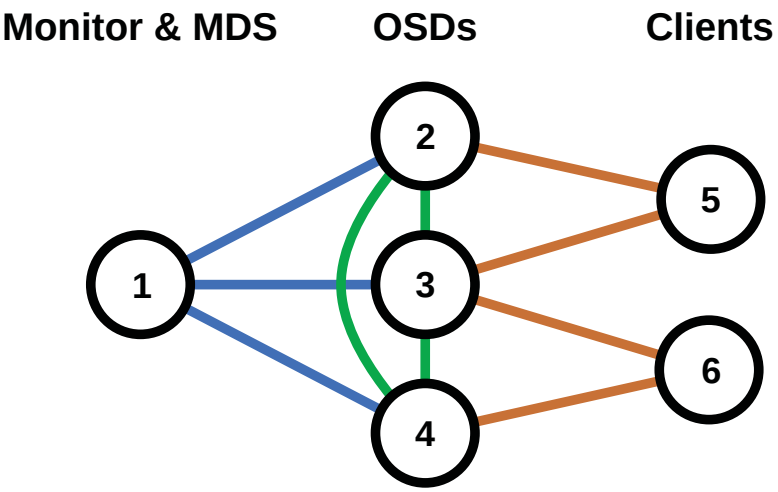


Figure 5.1: Abstract topology of the testbed, with 1 monitor node, 3 object storage devices, and 2 clients. The network zones are highlighted, with MDS-OSD (1–2, 1–3, and 1–4) communication in blue, OSD-OSD (2–3, 2–4, and 3–4) in green, and CLI (2–5, 3–5, 3–6, and 4–6) in orange.

All links in this testbed are rate-limited to 100 Mbps to provide a clear limit to the bandwidth. As there is no additional latency added in this stage, this configuration represents a small network that should not experience significant bandwidth limitations or latency.

The experiment was carried out by writing an arbitrary file from Node 5 into Ceph, then reading it back and verifying the hash. Timing for both read and write operations were recorded, and the transient throughput was monitored separately by periodically dumping the flow counters from the six switches. Total throughput can be confirmed by comparing the total traffic into and out of the Open vSwitch situated in Node 5.

To determine basic performance of the network when used without MPTCP, the functionality was turned off and the spanning-tree module (not the Spanning Tree Protocol) was run on top of POX to provide basic routing functionality. The results would be used as the baseline for comparison against the evaluations with smoc and an additional routing algorithm.

In the same manner as the experiments in the earlier chapters, smoc was de-

Table 5.1: Time to transfer each file in a simple write-read test using different algorithms and number of MPTCP subflows per port. Legend: S-T = Spanning-Tree (POX's implementation), smoc = algorithm in the original version of smoc, W = Time to write (s), R = Time to read (s)

	S-T		smoc	
File Size	W (s)	R (s)	W (s)	R (s)
5 GB	627.03	442.81	436.66	130.47
1 MB	0.35	0.37	4.27	4.14

ployed as the OpenFlow controller in the network and the experiments were repeated.

The algorithms to be experimented on includes smoc's original routing algorithm, which consists of one single shortest path followed by the least conflicting paths as secondary routes, as shown in Algorithm 1.

For the write/read test, the large file used is a 5 GB * file. The small files were 1 MB each, with 1000 files in total. All files are pseudo-randomly generated from `/dev/urandom` stream. File transfers are done five times to obtain the average time.

5.3 Experiment Results and Discussion

5.3.1 Performance of Write and Read operations (Client-OSD)

The times to transfer one file for each case are shown as in Table 5.1. The routing algorithm presented by smoc was considered to successfully fulfill its role as a multipath routing algorithm. In case of large files, smoc implementation performed significantly better, writing 43% and reading 240% faster†. The extraordinarily high data read speed increase could be related to the full parallelization of the read process when using MPTCP in conjunction with smoc. However, this is not

*For clarification purposes, prefixes use the SI system. In this case, 5 GB = 5×10^9 bytes.
†Based on $(B/A) - 1$ formula

Table 5.2: Data volume transferred by link and category when writing to DSS with (a) POX Spanning-Tree and (b) smoc. Amounts <1 MB and their sums are represented by "<". NL means no link. Amounts are in MB unless indicated.

(a) Writing with POX Spanning-Tree

From \ To	**Node 1**	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Node 6**
Node 1	-	<	<	<	NL	NL
Node 2	<	-	3.04	5.27 GB	27.85	NL
Node 3	<	3.06	-	3.93	<	<
Node 4	<	5.05 GB	3.35	-	NL	<
Node 5	NL	5.01 GB	<	NL	-	NL
Node 6	NL	NL	<	<	NL	-

Communication Category	**Volume**	**%**
MDS-OSD	<	<0.01%
OSD-OSD	10.33 GB	67.19%
CLI	5.04 GB	32.80%
Grand Total	15.48 GB	100%

(b) Writing with smoc

From \ To	**Node 1**	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Node 6**
Node 1	-	19.85	395.08	534.44	NL	NL
Node 2	793.87	-	677.33	684.19	52.05	NL
Node 3	13.03	5.03 GB	-	13.09	58.08	15.61
Node 4	142.39	23.36	721.33	-	NL	360.72
Node 5	NL	2.14 GB	3.19 GB	NL	-	NL
Node 6	NL	NL	360.72	15.61	NL	-

Communication Category	**Volume**	**%**
MDS-OSD	1.90 GB	12.46%
OSD-OSD	7.15 GB	46.91%
CLI	6.19 GB	40.63%
Grand Total	15.23 GB	100%

Table 5.3: Data volume transferred by link and category when reading from DSS with (a) POX Spanning-Tree and (b) smoc. Amounts <1 MB and their sums are represented by "<". NL means no link. Amounts are in MB unless indicated.

(a) Reading with POX Spanning-Tree

From \ To	**Node 1**	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Node 6**
Node 1	-	<	<	<	NL	NL
Node 2	<	-	1.81	1.59	5.01 GB	NL
Node 3	<	1.65	-	1.64	<	<
Node 4	<	1.53	1.68	-	NL	<
Node 5	NL	11.46	<	NL	-	NL
Node 6	NL	NL	<	<	NL	-

Communication Category	**Volume**	**%**
MDS-OSD	<	<0.01%
OSD-OSD	9.90	0.20 %
CLI	5.02 GB	99.80%
Grand Total	5.03 GB	100%

(b) Reading with smoc

From \ To	**Node 1**	**Node 2**	**Node 3**	**Node 4**	**Node 5**	**Node 6**
Node 1	-	1.61 GB	435.50	35.26	NL	NL
Node 2	23.99	-	1.30 GB	25.29	2.29 GB	NL
Node 3	11.28	606.48	-	22.83	3.03 GB	11.49
Node 4	2.05 GB	1.39 GB	1.22 GB	-	NL	663.03
Node 5	NL	33.71	59.72	NL	-	NL
Node 6	NL	NL	663.03	11.49	NL	-

Communication Category	**Volume**	**%**
MDS-OSD	4.17 GB	26.88%
OSD-OSD	4.57 GB	29.48%
CLI	6.77 GB	43.64%
Grand Total	15.51 GB	100%

the case for writing. Write operations with multiple agents must be done carefully with consistency in mind. While many systems can work on an "eventually consistent" basis, Ceph as a DSS intended to store data is not designed to work with such inconsistencies, and would rather limit writing to only one copy by one user at a time rather than allowing file updates to be written to many physical locations and relying on synchronization.

Unsurprisingly, as explained in Sect. 4.4.2, smoc performs comparatively worse than POX S-T for the short file cases due to the seconds-long path installation delay. This is inherent to more complex OpenFlow implementations. Since real-world DCNs have more permanence in terms of network configuration, it is advisable to have production systems use a more permanent and encompassing way to install flow rules. Due to the emphasis of this work on large file transfers, communication pattern of large files was also observed, collected, and analyzed.

5.3.2 Observation of Communication Patterns

Communication in DSSes can be roughly categorized into three broad groups. The first group is the client communication, which includes the metadata interaction between the client and the MDS, and also the actual data transfer between the client and the OSDs. The second group is the data transfer between the OSDs (OSD-OSD) for data replication, system recovery, and synchronization of updates to stored objects. The third group consists of the internal interaction between the MDS and the OSDs (MDS-OSD), which can include important housekeeping messages like heartbeats and file mapping updates. The categorization applied to this work is visualized in Fig. 5.1.

All three groups of communication have unique requirements, specific characteristics, and are handled differently. In practical implementations, client communication is usually restricted to a public network. Ceph documentation also recommends that system architects build their DSSes with separate private and public networks. OSD-OSD communication may require large bandwidth and occur in large bursts. Finally, the MDS-OSD interaction ensures the continuity of the system as the cessation of these messages can result in an inconsistent view of the system and therefore instability. Both OSD-OSD and MDS-OSD communication groups are usually confined to the private network.

With these considerations in mind, the obtained data was more thoroughly explored. By periodically collecting port statistics of the switches in the network, it was possible to create a matrix of data volume sent between each pair of hosts (as shown in Table 5.2 and Table 5.3) and determine where the communication occurs the most, and visualize it as a graph as shown in Fig. 5.2 and Fig. 5.3. Please note that these are not averages of five experimental replications like the time measurement. This is because the routing and congestion control behaviors can differ between runs, complicating the aggregate data. Instead, the result from a single experimental replication was chosen based on the clarity and typicality of the communication pattern represented.

When using POX Spanning-Tree to write or read the data, network traffic is largely confined into a single line in case of large file transfer. However, when using smoc for the same tasks, network traffic is more evenly spread throughout the network, sometimes even into the MDS-OSD area. Before using smoc, traffic in the MDS-OSD area was negligible for both writing and reading. This can be seen either as a risk or a benefit depending on the design and view point of the operator. However, for the purposes of this work, this is regarded as an improvement to path diversity, which improves the resilience of data transfer.

Network traffic into and out of the client was also more balanced. The share of data communication increased to 12.46% and 26.88% for write and read operations. Balance between two interfaces of the client was also observed, as without smoc all 5 GB of traffic occurred through only one interface, but was spread to a more equal share of 2 GB + 3 GB after introducing the controller.

Write Pattern of POX Spanning-Tree and smoc

When writing data to the DSS, POX Spanning-Tree directed virtually all data through only one interface. This is reflected in the "Node 5" row in Table 5.2a where the node sends almost the entirety of its data towards Node 2, while sending a negligible amount to Node 3. The bulk of the communication occurred between the two OSD nodes, Node 2 and 4.

By inspecting the "heatmap" plots as shown in Fig. 5.2 and the transient plots based on the communication category in Fig. 5.4, the replication of data between the OSDs occur concurrently with the transmission of data from the client to one

of the OSDs. This concurrent replication behavior could be one of the factors that contribute to a lower throughput gain when using smoc to write, compared to read. There was significant communication in the MDS-OSD area that appeared when smoc was in use, but not when POX Spanning-Tree was used, suggesting that smoc directed some of the traffic through that area.

By inspecting the interface-based plot in Fig. 5.6, it is apparent that for these specific cases, using smoc with MPTCP gives the node the proper ability to communicate using both interfaces. Due to the recent implementation of new MPTCP congestion control, however, MPTCP became less aggressive and essentially abandoned one of the interfaces during the latter half of the data transfer illustrated in Fig. 5.6b. The facts that total throughput (about 50 Mbps, based on number about 20 MB per 3 seconds in the graph) remained the same, not reduced to half (as shown earlier in Sect. 3.3.5), and did not reach the upper aggregate limit (2×100 Mbps) suggest that the limiting factor could be related to I/O, OS, physical storage devices, or other environmental operations beyond the scope of the controller. Still, the graphs and the heatmaps showed that the paths taken became more diverse as MPTCP and smoc were implemented.

Both cases exhibited at least 5 GB of client communication, and also at least 5 GB of OSD-OSD communication. Based on the architecture of Ceph outlined in the documentation [‡], the OSD-OSD communication could be explained as replication operation. Since the communication pattern may be different based on the DSS and their configurations, using other DSS, for example HDFS or GFS, would result in a different communication pattern and therefore the performance gains as well.

Read Pattern of POX Spanning-Tree and smoc

It was unsurprising that smoc performed better than POX when reading the large file due to the larger bandwidth made available by MPTCP. The measured times were much lower for the read compared to the write operations, due to the lack of external limiting factors discussed in the previous section. As indicated in Table 5.3 and reflected in Fig. 5.3, there was a larger path diversity when using smoc with MPTCP, including the go-around paths that involve the MDS-OSD

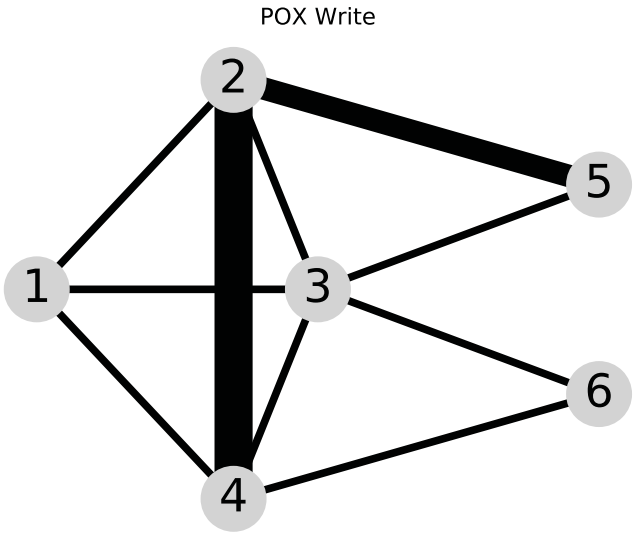
[‡] `http://docs.ceph.com/docs/jewel/architecture/`

area of the network. While this might be seen as counterproductive, such usage will be automatically eliminated by MPTCP congestion control when sufficient congestion is present in the network or a path that has a clearly better performance is decided. This behavior was also equally shown in transient plots shown in Fig. 5.5 and Fig. 5.7, where more diversity both in communication areas and the activity of client interfaces are observed when using smoc compared to POX Spanning-Tree.

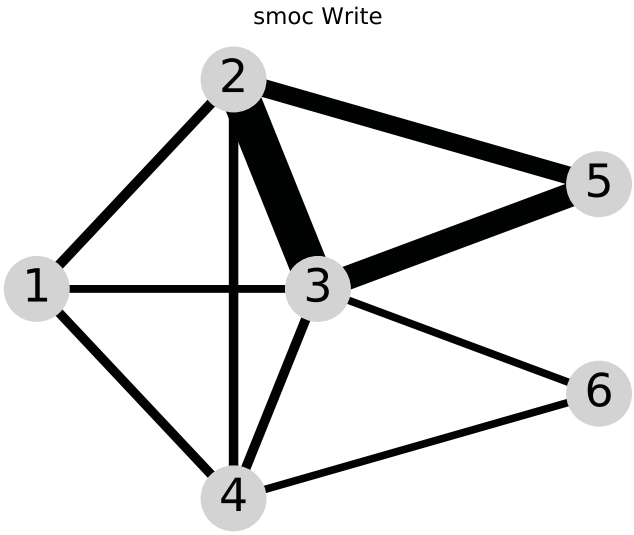
5.3.3 Additional Observations

In addition to the slower small-file operations, an additional simple test was performed to test the time for no-file operations as a basis for short flows between the client and the servers. In particular, an empty directory list of Ceph objects in an empty object storage pool was acquired five times in succession. The average time to execute no-file operations with smoc running MPTCP was 8.192 s with standard deviation of 0.02501 s (CV = 0.3053%), suggesting that the time is relatively constant. As with the previous experiments, this could be caused by Overseer and POX's housekeeping functions that add delay to the installation of flow entries, causing all commands to be delayed.

Furthermore, using MPTCP with POX Spanning-Tree controller resulted in errors. This is understandable because basic POX modules could not understand the notion of having a network interface appear from multiple ingress ports, causing the controller to believe that an IP or MAC address has relocated. This further establishes the importance of providing a properly implemented multipathing-aware controller when using any OpenFlow network and striving to take advantage of having multiple paths.

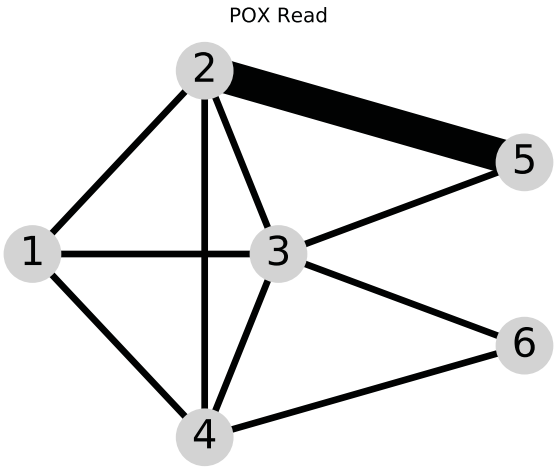


(a) Writing with POX Spanning-Tree

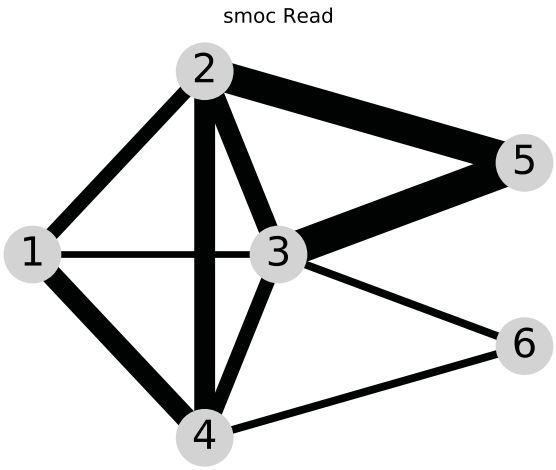


(b) Writing with smoc

Figure 5.2: Visualization of relative data transfer volume when writing to the DSS, based on data in Table 5.2a and Table 5.2b, as undirected graphs

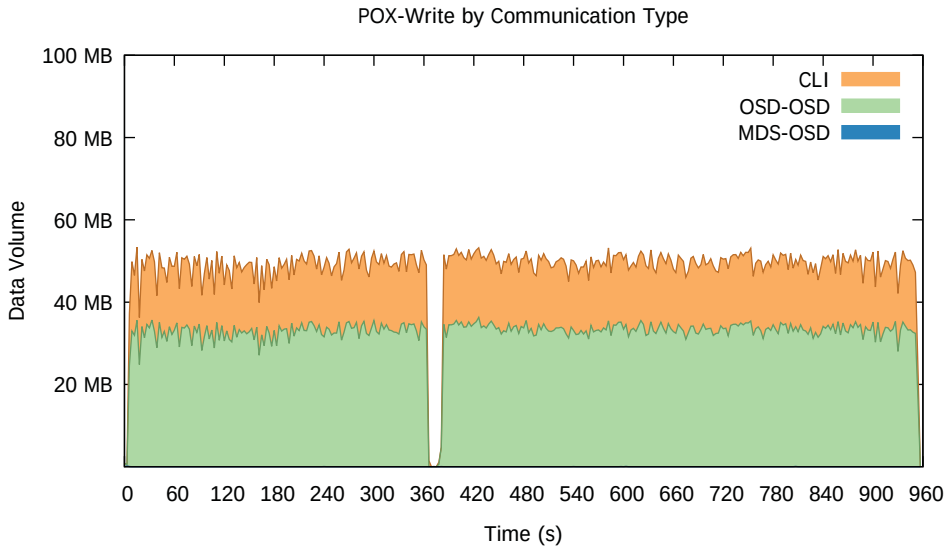


(a) Reading with POX Spanning-Tree

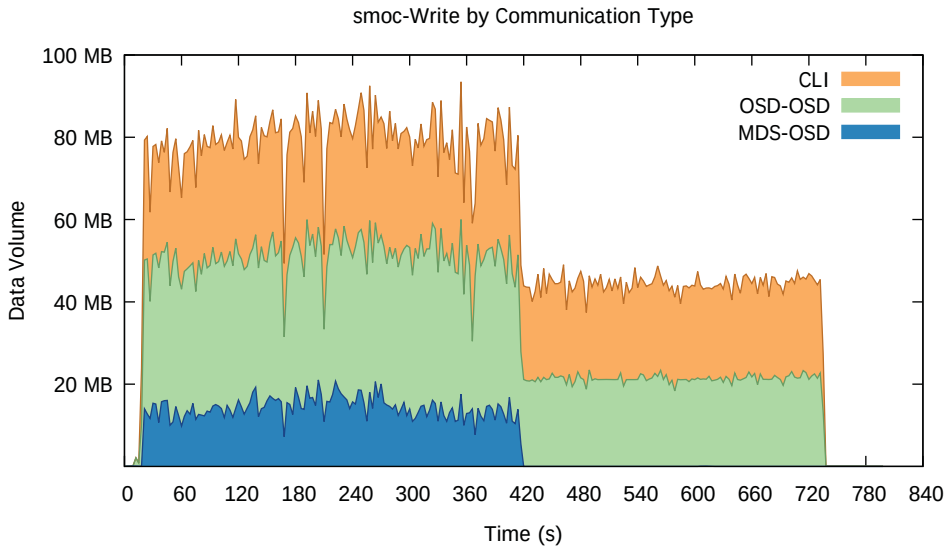


(b) Reading with smoc

Figure 5.3: Visualization of relative data transfer volume when reading from the DSS, based on data in Table 5.3a and Table 5.3b, as undirected graphs

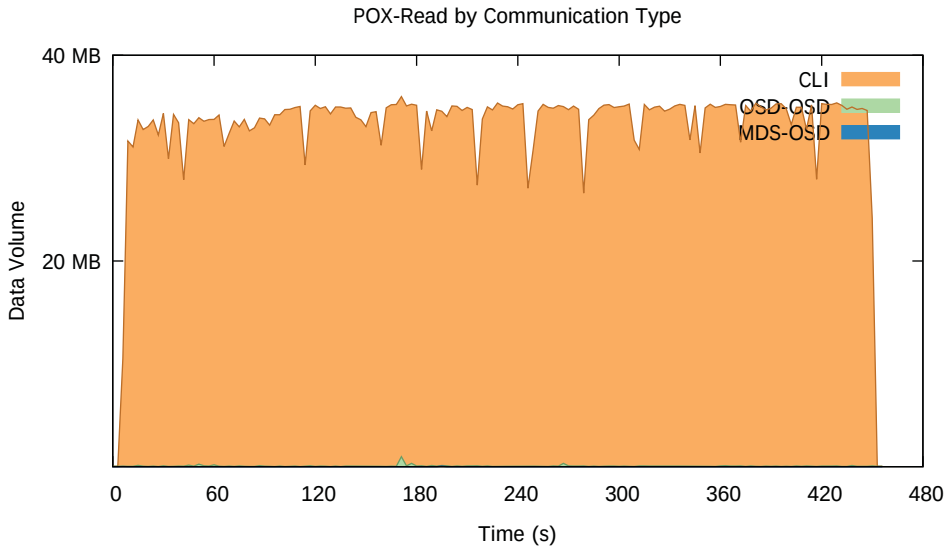


(a) Writing with POX Spanning-Tree

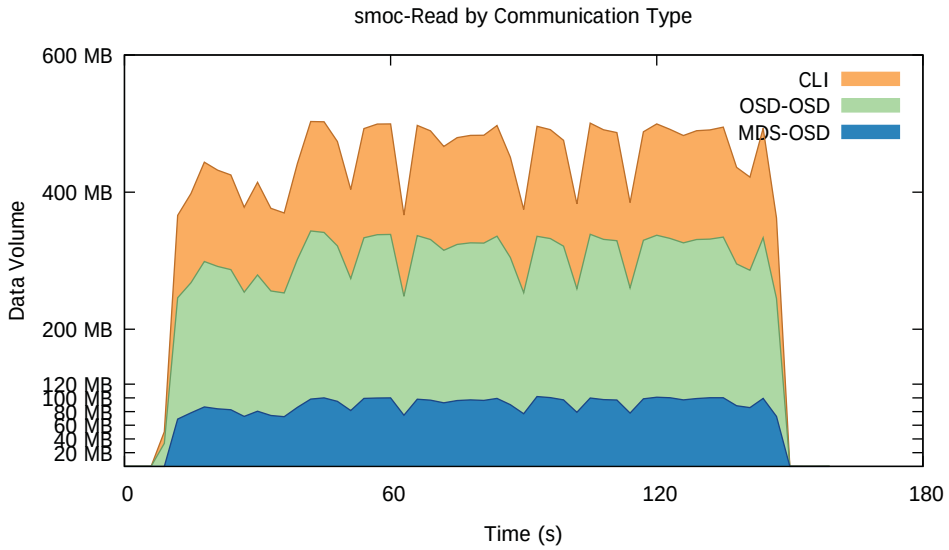


(b) Writing with smoc

Figure 5.4: Transient stacked area plot of aggregate throughput by communication types when writing a large file to the DSS.

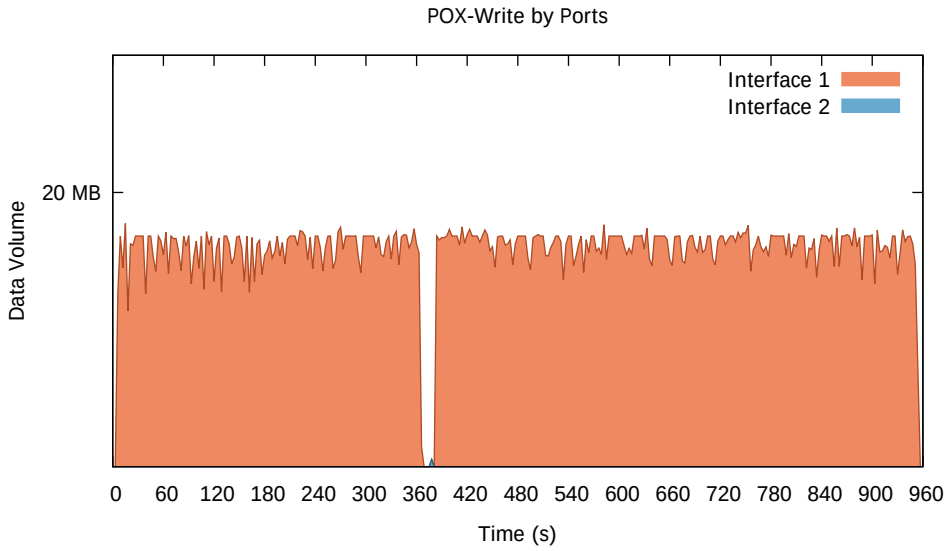


(a) Reading with POX Spanning-Tree

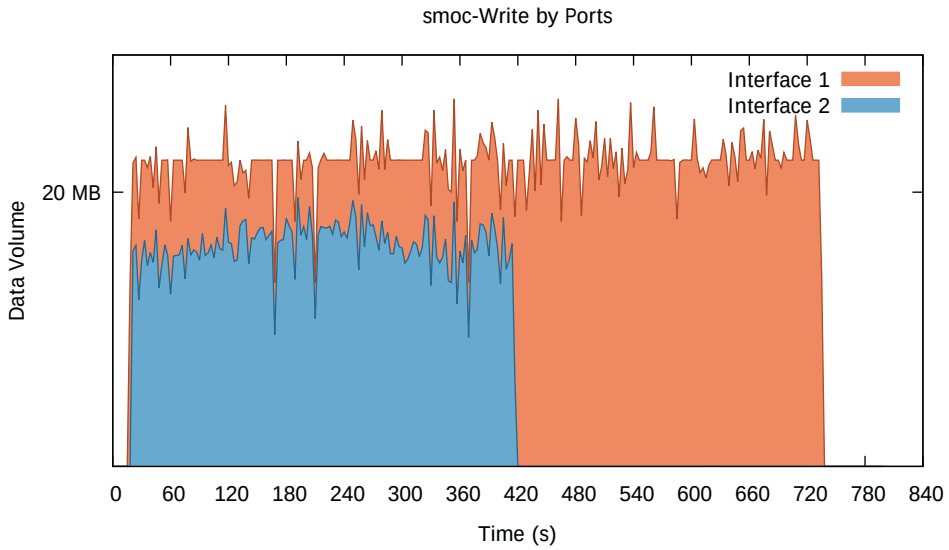


(b) Reading with smoc

Figure 5.5: Transient stacked area plot of aggregate throughput by communication types when reading a large file from the DSS.

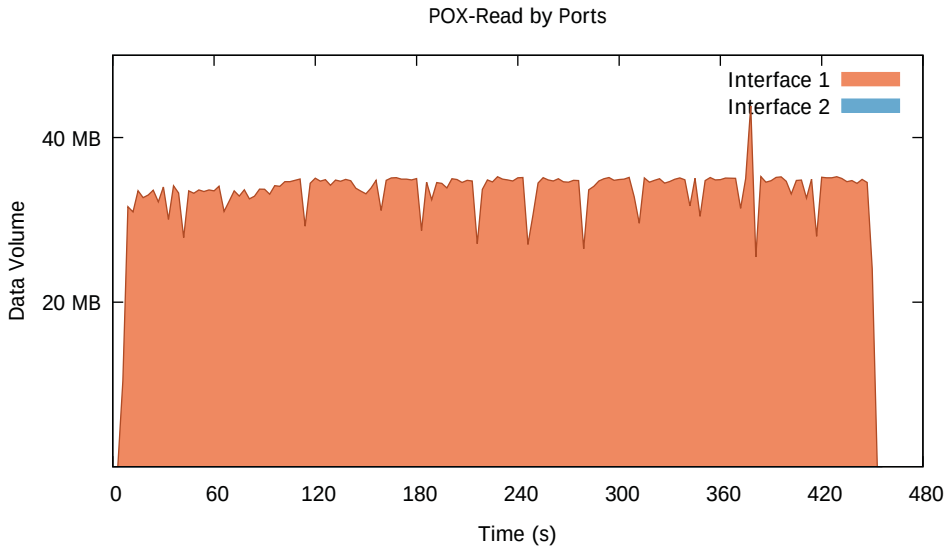


(a) Writing with POX Spanning-Tree

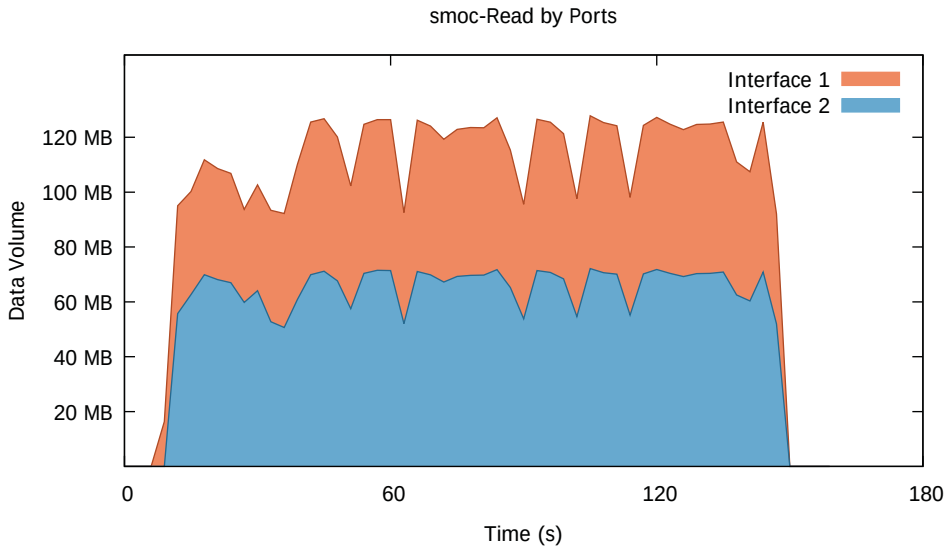


(b) Writing with smoc

Figure 5.6: Transient stacked area plot of aggregate throughput sent from the interfaces of Node 5 (the client) when writing a large file to the DSS.



(a) Reading with POX Spanning-Tree



(b) Reading with smoc

Figure 5.7: Transient stacked area plot of aggregate throughput received by the interfaces of Node 5 (the client) when writing a large file to the DSS.

6 Conclusion

In this dissertation, the Multipath TCP performance in an OpenFlow network was explored. An MPTCP-aware network controller, smoc, was developed to address the problem of path conflicts that arise when attempt to route multipath traffic through a network that is not multipath-aware. Furthermore, the controller was tested against a deployment of Ceph distributed storage system to determine write and read performance gain when using smoc and MPTCP compared to using basic shortest-path and Spanning-Tree controller.

The primary feature of the controller was its ability to quickly detect and identify MPTCP flow-subflow relationship which is the quality that sets it apart from the MPTCP-unaware controllers or ordinary routers. This was made possible by monitoring the data plane of the network for the initiation of TCP streams, detecting if they are MPTCP, and matching them against existing connections into a group as shown in Algorithm 3. From this point, any action could be taken on these flows, as demonstrated in Chap. 4 where the traffic was routed according to the original smoc routing algorithm implemented in Algorithm 1.

Based on the results of the experiments, the leverage provided by the controller, and the efficiency of the proposed algorithm, there should be a case for Multipath TCP should be widely adopted in the future when these algorithms are more thoroughly tested and verified. However, current research on multipath networking concerns mostly wireless networks. Attention to bring MPTCP into high-performance computing as a potentially efficient backwards-compatible algorithm, along with using OpenFlow to provide a customizable routing algorithm to the multipath traffic could lead to a better performance in DSSes and distributed systems in general.

Since the methods presented in this work does not include the monitoring of bandwidth information, it is recommended that network metrics should also be

considered when working with network routing. However, it is important to note that bandwidth and latency monitoring can add load to the system, and it is possible to also adversely affect the performance in DCNs due to their already low and constant latency. Nevertheless, smoc showed that even the most basic of routing mechanisms could be used to reliably route MPTCP traffic. The choice of routing protocols or network metrics doesn't change this fact: as long as MPTCP subflow groups could be identified, a fully MPTCP-aware network controller would still largely work, regardless of the finer details in network configuration.

As a strictly bare-bones controller designed to implement only the core mechanisms necessary for recognizing and routing MPTCP traffic, smoc has achieved its purposes. It is distinct from general-purpose SDN controllers that usually work on either bandwidth-based or metric-based routing. The treatment of different MPTCP sessions has also, up to date of first published paper within this dissertation, been a novel approach. Finally, the topological routing algorithm in this work guarantees that it will always, bandwidth wise and unless there are circumstances complicating MPTCP congestion control, perform equally well or better than single-path algorithms and spanning trees.

In summary, it is demonstrated in this work that a combination of OpenFlow and Multipath TCP could be used to potentially utilize the full amount of bandwidth whenever multiple paths are available between a pair of hosts in the network. This work also serves as a step towards the future Internet, where more bandwidth could be utilized to accommodate the ever-growing demands of the various applications.

Acknowledgements

I wish to thank the following people for their guidance, support, and assistance. Without these people, this work would not have been possible.

Firstly, I would like to express my appreciation to my research supervisor, Professor Hajimu Iida for his support, early financial assistance, work assignment, and valuable feedback. His support is of absolute importance for my life in Japan.

Secondly, I would like to thank Professor Kazutoshi Fujikawa for taking reviewing, discussing, and providing feedback for my work, without which the work would not have been complete as it is.

I wish to thank my advisor, Associate Professor Kohei Ichikawa. When I decided I would be joining NAIST, I had no experience with research or even the school itself. It obviously takes a lot of trust to support someone one barely knows, but he did confide in my ability and potential. He provided a lot of guidance, experience, support, and opportunities.

I express additional thanks to Assistant Professor Putchong Uthayopas for his guidance during my undergraduate years. In fact, he was the one who introduced me to high-performance computing when I first met him during high school. After joining his laboratory, he provided rigorous instruction on research methodology and guidance on my future career. He also introduced me to the Pacific Rim Applications and Grid Middleware Assembly or PRAGMA.

I also wish to extend my gratitude to the professors of Laboratory for Software Design and Analysis (SDLAB), as they provided me with a lot of opportunity and guidance. Associate Professor Toshinori Takai introduced me to the IT-Triadic Program at NAIST and provided me with opportunities to collaborate with and visit renowned institutions such as the JAXA and Nagoya University. Associate Professor Norihiro Yoshida and Assistant Professor Ana Erika Camargo Cruz provided additional feedback and comments during my working on Master's thesis,

while Assistant Professor Yasuhiro Watashiba and Assistant Professor Eunjong Choi provided further assistance during the latter years as a doctoral student. Although visiting SDLAB for a limited time, Associate Professor Daniel Port from Shidler College of Business, University of Hawai'i at Manōa provided new insights that could not be obtained otherwise. I wish to express my appreciation to everyone at Creative and International Competitiveness Project (CICP) and Global Entrepreneur in Internet of Things (GEIOT) program, with which I have earned significant opportunities to explore and educate.

To all the good people of PRAGMA: Dr. Peter Arzberger, Associate Professor Philip M. Papadopoulos, Ms. Teri Simas, Ms. Nadya Williams, Ms. Shava Smallen, and Mr. Luca Clementi provided me with a lot of assistance, advice, inspiration, and instruction while I was interned at University of California, San Diego in 2014. I also express my gratitude towards Professor Beth A. Plale for her guidance to PRAGMA Students Steering Committee.

The JASSO Honors Scholarship provided me with financial support during FY 2013, followed by the MEXT Top Global University Project Scholarship in FY 2014, KDDI Foundation Scholarship for FY 2015, and SGH Foundation Scholarship since FY 2016. I feel honored to be part of these families, and wish to express my gratitude to all of these organizations.

To Friendship, I also express my thanks to all members of the Laboratory for Software Design and Analysis, especially my tutor and groupmates who actively provided assistance and feedback for my research. To PRAGMA Students Steering Committee including but not limited to "Class of 2018" Pongsakorn U-Chupala, Quan Zhuo, and Meilan Jiang, as well as alumnus Yuan Luo and newcomers Wassapon Watanakeesuntorn, Can Wu and Giljae Lee. To Thai friends in Nara; and to new friends from Ingress Resistance Kansai club.

To Patcha Yanpirat, for her support, care, and suggestions, some of which the most important I have ever received. Even "I will give you everything I have" would still not suffice as a proper reciprocation of this kindness.

Last but not least, I wish to express my highest gratitude to my dearest parents for the most precious gift of life, raising and educating me with great care, and instilling the values of education and ethics since my youth. No amount of words would sufficiently express my gratitude. I thank both my parents for everything.

Bibliography

[1] William Allcock et al. “Protocols and services for distributed data-intensive science”. In: *AIP Conference Proceedings*. IOP INSTITUTE OF PHYSICS PUBLISHING LTD. 2000, pp. 161–163. URL: http://globusproject.org/alliance/publications/papers/ACAT3.pdf.

[2] William Allcock et al. “The Globus striped GridFTP framework and server”. In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society. 2005, p. 54. URL: https://www.dc.uba.ar/materias/aerg/2006/cuat2/downloads/gridftp-final.pdf.

[3] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*. RFC 5681. RFC Editor, Sept. 2009.

[4] Sébastien Barré et al. “Experimenting with Multipath TCP”. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM ’10. New Delhi, India: ACM, 2010, pp. 443–444. ISBN: 978-1-4503-0201-2. DOI: 10.1145/1851182.1851254. URL: http://doi.acm.org/10.1145/1851182.1851254.

[5] Peter J Braam and Rumi Zahir. “Lustre: A scalable, high performance file system”. In: *Cluster File Systems, Inc* (2002).

[6] Neal Cardwell et al. “BBR: Congestion-based congestion control”. In: *Queue* 14.5 (2016), p. 50.

[7] Bachir Chihani and Denis Collange. *A Survey on Multipath Transport Protocols*. http://xxx.tau.ac.il/pdf/1112.4742.pdf. arXiv: 1112.4742v1.

[8] Andrew R Curtis et al. “DevoFlow: Scaling flow management for high-performance networks”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 254–265.

[9] Alex Davies and Alessandro Orsaria. “Scale out with GlusterFS”. In: *Linux J.* 2013.235 (Nov. 2013). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=2555789.2555790.

[10] Yu Dong et al. “Multi-Path Load Balancing in Transport Layer”. In: *3rd EuroNGI Conference on Next Generation Internet Networks*. May 2007, pp. 135–142. DOI: 10.1109/NGI.2007.371208.

[11] Marcial P Fernandez. “Comparing openflow controller paradigms scalability: Reactive and proactive”. In: *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE. 2013, pp. 1009–1016.

[12] A. Ford et al. *Architectural Guidelines for Multipath TCP Development*. RFC 6182. RFC Editor, Mar. 2011.

[13] A. Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. RFC Editor, Jan. 2013.

[14] Bryan Ford and Janardhan Iyengar. “Breaking up the transport logjam”. In: *ACM HotNets, October*. 2008.

[15] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. In: *SIGOPS Oper. Syst. Rev.* 37.5 (Oct. 2003), pp. 29–43. ISSN: 0163-5980. DOI: 10.1145/1165389.945450. URL: http://doi.acm.org/10.1145/1165389.945450.

[16] Dan Gunter et al. “Exploiting Network Parallelism for Improving Data Transfer Performance”. In: *2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*. IEEE. 2012, pp. 1600–1606.

[17] 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: *Proceedings of the 16th International Parallel and Distributed Processing Symposium*. IPDPS ’02. IEEE. 2001, 10–pp. URL: http://www-personal.umich.edu/~hacker/IPDPS.pdf.

[18] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Laboratory (LANL), 2008.

[19] B. Hesmans et al. "SMAPP: Towards Smart Multipath TCP-enabled Applications". In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT '15. Heidelberg, Germany: ACM, 2015, 28:1–28:7. ISBN: 978-1-4503-3412-9. DOI: 10.1145/2716281.2836113. URL: http://doi.acm.org/10.1145/2716281.2836113.

[20] C. Hopps. *Analysis of an Equal-Cost Multi-Path Algorithm*. RFC 2992. RFC Editor, Nov. 2000.

[21] Che Huang et al. "A multipath controller for accelerating GridFTP transfer over SDN". In: *11th IEEE International Conference on eScience*. Munich, Germany. Sept. 2015, pp. 439–447.

[22] Che Huang et al. "A Multipath OpenFlow Controller for GridFTP". In: *The 1st. cross-disciplinary Workshop on Computing Systems, Infrastructures, and Programming* (Apr. 2017).

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

[24] Kohei Ichikawa et al. "PRAGMA-ENT: An International SDN testbed for cyberinfrastructure in the Pacific Rim". In: *Concurrency and Computation: Practice and Experience* 29.13 (2017). Previously presented at PRAGMA Workshop on International Clouds for Data Science 2015, Best Paper Award., e4138–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.4138.

[25] V. Jacobson. "Congestion Avoidance and Control". In: *Symposium Proceedings on Communications Architectures and Protocols*. SIGCOMM '88. Stanford, California, USA: ACM, 1988, pp. 314–329. ISBN: 0-89791-279-9. DOI: 10.1145/52324.52356. URL: http://doi.acm.org/10.1145/52324.52356.

[26] Nick McKeown et al. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[27] Chawanat Nakasan, Kohei Ichikawa, and Putchong Uthayopas. “Performance Evaluation of MPTCP over OpenFlow Network”. In: *IPSJ SIG Notes 2014-HPC-145*. Vol. 145. 30. Information Processing Society of Japan (IPSJ), July 2014, pp. 1–6. URL: http://ci.nii.ac.jp/naid/110009808125/en/.

[28] Chawanat Nakasan et al. “A Simple Multipath OpenFlow Controller using topology-based algorithm for Multipath TCP”. In: *PRAGMA Workshop on International Clouds for Data Science (PRAGMA-ICDS 2015)*. Depok, Indonesia. Oct. 2015.

[29] Chawanat Nakasan et al. “A simple multipath OpenFlow controller using topology-based algorithm for multipath TCP”. In: *Concurrency and Computation: Practice and Experience* 29.13 (2017), e4134–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.4134.

[30] *Open vSwitch*. https://github.com/openvswitch/ovs. 2009.

[31] Christoph Paasch, Gregory Detal, and David Heidelberger. *Multipath TCP*. https://github.com/multipath-tcp. 2014.

[32] Christoph Paasch et al. “Exploring Mobile/WiFi Handover with Multipath TCP”. In: *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design*. CellNet ’12. Helsinki, Finland: ACM, 2012, pp. 31–36. ISBN: 978-1-4503-1475-6. DOI: 10.1145/2342468.2342476. URL: http://doi.acm.org/10.1145/2342468.2342476.

[33] Martin Placek and Rajkumar Buyya. “A taxonomy of distributed storage systems”. In: *Technical Report, The University of Melbourne, The Cloud Computing and Distributed Systems (CLOUDS) Laboratory* (2006).

[34] Ronald van der Pol et al. “Multipathing with MPTCP and OpenFlow”. In: *2012 SC Companion: High Performance Computing, Networking, Storage and Analysis (SCC)*. IEEE. 2012, pp. 1617–1624.

[35] Costin Raiciu et al. “How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 29–29. URL: http://dl.acm.org/citation.cfm?id=2228298.2228338.

[36] Ioan Raicu, Ian T. Foster, and Pete Beckman. “Making a Case for Distributed File Systems at Exascale”. In: *Proceedings of the Third International Workshop on Large-scale System and Application Performance*. LSAP '11. San Jose, California, USA: ACM, 2011, pp. 11–18. ISBN: 978-1-4503-0703-1. DOI: 10.1145/1996029.1996034. URL: http://doi.acm.org/10.1145/1996029.1996034.

[37] Kultida Rojviboonchai and AIDA Hitoshi. “An evaluation of multi-path transmission control protocol (M/TCP) with robust acknowledgement schemes”. In: *IEICE transactions on communications* 87.9 (2004), pp. 2699–2707.

[38] *Scientic Expeditions - PRAGMA*. http://www.pragma-grid.net/expeditions.php. Accessed: 2015-02-05.

[39] C. E. Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37.1 (Jan. 1949), pp. 10–21. ISSN: 0096-8390. DOI: 10.1109/JRPROC.1949.232969.

[40] Konstantin Shvachko et al. “The hadoop distributed file system”. In: *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. IEEE. 2010, pp. 1–10.

[41] T. Socolofsky and C. Kale. *A TCP/IP Tutorial*. RFC 1180. RFC Editor, Jan. 1991.

[42] Sun Microsystems, Inc. *NFS: Network File System Protocol Specification*. RFC 1094. RFC Editor, Mar. 1989.

[43] Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. “Gfarm Grid File System”. In: *New Generation Computing* 28.3 (July 2010), pp. 257–275. ISSN: 1882-7055. DOI: 10.1007/s00354-009-0089-5. URL: https://doi.org/10.1007/s00354-009-0089-5.

[44] D. Thaler and C. Hopps. *Multipath Issues in Unicast and Multicast Next-Hop Selection*. RFC 2991. RFC Editor, Nov. 2000.

[45] Tran Doan Thanh et al. "A Taxonomy and Survey on Distributed File Systems". In: *2008 Fourth International Conference on Networked Computing and Advanced Information Management*. Vol. 1. Sept. 2008, pp. 144–149. DOI: 10.1109/NCM.2008.162.

[46] Amin Tootoonchian and Yashar Ganjali. "HyperFlow: A distributed control plane for OpenFlow". In: *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association. 2010, pp. 3–3.

[47] Pongsakorn U-chupala et al. "Application-Oriented Bandwidth and Latency Aware Routing with OpenFlow Network". In: *6th IEEE International Conference on Cloud Computing Technology and Science*. 2014. ISBN: 5314550091.

[48] Pongsakorn U-chupala et al. "Designing of SDN-Assisted Bandwidth and Latency Aware Route Allocation". In: *Summer United Workshops on Parallel, Distributed and Cooperative Processing (SWoPP)*. 2014.

[49] Sage A. Weil et al. "Ceph: A Scalable, High-Performance Distributed File System". In: *Presented as part of the 7th USENIX Symposium on Operating Systems Design and Implementation*. Seattle, WA: USENIX, 2006. URL: https://www.usenix.org/conference/osdi-06/ceph-scalable-high-performance-distributed-file-system.

[50] Sage A Weil et al. "CRUSH: Controlled, scalable, decentralized placement of replicated data". In: *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM. 2006, p. 122.

[51] Sage A. Weil et al. "RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters". In: *Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*. PDSW '07. Reno, Nevada: ACM, 2007, pp. 35–44. ISBN: 978-1-59593-899-2. DOI: 10.1145/1374596.1374606. URL: http://doi.acm.org/10.1145/1374596.1374606.

[52] D. Zhou et al. “Multipath TCP for user cooperation in LTE networks”. In: *IEEE Network* 29.1 (Jan. 2015), pp. 18–24. ISSN: 0890-8044. DOI: 10.1109/MNET.2015.7018199.

Publication List

Kohei Ichikawa et al. “PRAGMA-ENT: An International SDN testbed for cyberinfrastructure in the Pacific Rim”. In: *Concurrency and Computation: Practice and Experience* 29.13 (2017). Previously presented at PRAGMA Workshop on International Clouds for Data Science 2015, Best Paper Award., e4138–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.4138.

Chawanat Nakasan, Kohei Ichikawa, and Putchong Uthayopas. “Performance Evaluation of MPTCP over OpenFlow Network”. In: *IPSJ SIG Notes 2014-HPC-145*. Vol. 145. 30. Information Processing Society of Japan (IPSJ), July 2014, pp. 1–6. URL: http://ci.nii.ac.jp/naid/110009808125/en/.

Chawanat Nakasan et al. “A Simple Multipath OpenFlow Controller using topology-based algorithm for Multipath TCP”. In: *PRAGMA Workshop on International Clouds for Data Science (PRAGMA-ICDS 2015)*. Depok, Indonesia. Oct. 2015.

Chawanat Nakasan et al. “A simple multipath OpenFlow controller using topology-based algorithm for multipath TCP”. In: *Concurrency and Computation: Practice and Experience* 29.13 (2017), e4134–n/a. ISSN: 1532-0634. DOI: 10.1002/cpe.4134.