IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 26, 2018, accepted December 17, 2018, date of publication December 27, 2018,
date of current version January 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889868

Gate Switch Selection for In-Band Controlling in Software Defined Networking

ALI RAZA AND SANGHWAN LEE
Department of Computer Science, Kookmin University, Seoul 02707, South Korea

Corresponding author: Sanghwan Lee (sanghwan@kookmin.ac.kr)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2016R1D1A1B03934529.

ABSTRACT In software defined networking (SDN), control connections between switches and the controller are established by either an in-band or an out-of-band control mechanism. In this paper, we focus on in-band controlling. In an in-band controlling, a switch constructs a control connection (usually TCP connection) to the controller through a data path. To ensure path availability, better load balancing, and robustness, multi-path TCP (MPTCP) has been introduced for control channels instead of TCP. Since MPTCP has multiple subflows, each subflow may acquire different paths to fully utilize MPTCP for control connection of SDN. Essentially, the controller should be connected to multiple gate switches first and then, each non-gate switch fabricates an MPTCP connection with multiple subflows to the controller through the gate switches. In such MPTCP-based in-band controlling environment, the selection of gate switches is crucial in achieving availability and reliability of the control channel. For example, multiple subflows through the chosen gate switches should be disjoint as much as possible. In this paper, we first illustrate an objective of in-band controlling and then, we propose a heuristic algorithm to compute a gate switch set that achieves the objective. To be specific, we want to make the subflows disjoint and short lengthed for high availability. We investigate the performance of the proposed algorithm with the two baseline algorithms, i.e., *exhaustive search* and *random search*. Through extensive simulation, we demonstrate that the proposed algorithm performs much better than the random search and is comparable to the exhaustive search.

INDEX TERMS SDN, MPTCP, in-band controlling, gate switch selection, robustness of control channel.

I. INTRODUCTION

Traditional IP networks have become very complicated. Similarly, they are also difficult to reconfigure due to the diversity among vendor-specific networking devices [1]. Furthermore, both the data and control plane are vertically integrated inside networking devices. Thus, it is troublesome to deploy new networking architecture into the existing IP networks. To foster rapid deployment of a new networking architecture, Software Defined Networking (SDN) has been introduced as a new networking paradigm since last decade. Most network vendors are frequently adopting SDN in the hope to make their networks more flexible and manageable.

In SDN, the control and data plane are decoupled from each other. The SDN controller manages the control plane and the underlying networking devices (or switches) handle the data plane. Thus, the switches have become naturally the forwarding elements without any control plane functionality. One way to instantiate SDN is to use OpenFlow, which is the most well-known protocol between the controller and the switches [2]. Controllers can add, modify, and remove the flow entries of the switches by utilizing the OpenFlow messages. Thus, network operators can design applications operating on top of the SDN controller, which can efficiently configure, manage, and control the underlying network [3].

Due to this decoupling of the control and data planes, the reliability of the control channels between the controller and switches are fundamental for the appropriate operation of the network. If the control channel is not operating due to some reasons, the network devices may not transmit the packets through appropriate paths. In SDN, the control connection can be established either in in-band manner or out-of-band fashion. Unlike the out-of-band mechanism, in in-band situation, the control connection may not work properly due to the congestion of the data links or link failures. Especially, in OpenFlow protocol, the control connection between the controller and a switch is commonly a TCP connection. So the

TCP connection may be terminated when the underlying path is not available.

To overcome this problem, we have proposed an MPTCP based in-band controlling mechanism [4]. Basically, the controller and a switch constitute an MPTCP connection with multiple disjoint subflows. By adopting MPTCP in control connections, we can get three-folded benefits: link failure resiliency of the control connection, less delay of OpenFlow control messages, and higher bandwidth for control traffic in SDN network [5], [6]. For the MPTCP based control connection, we have designed the algorithms that select two gate switches [4]. To be precise, we have defined the problem of selecting a set of two gate switches, through which other switches establish an in-band control connection with the controller. The proposed algorithms work appropriately in selecting two gate switches and the resulting subflow paths from each switch are disjoint paths. However, the previous work lacks in completeness and performance. First, the algorithms accomplish for only selecting two gate switches, not more than two. Second, it is not apparent that how to define the disjointedness when the number of gate switches is more than two because some switches may have the smaller number of outgoing links than the number of gate switches. In this case, we cannot adequately utilize the chosen gate switches. Finally, we claim that the computation time of the heuristic algorithms can be lessened further.

To refine and intensify our previous work, in this paper, we propose an algorithm to select an *arbitrary* number of gate switches set. Furthermore, we introduce a practical definition of disjointedness called "k-partially disjoint" that can be used in an in-band based controlling. Basically, we denominate a set of k paths as the k-partially disjoint paths when a path is the complete disjoint with any of the other $k-1$ paths. Through simulation experiments, we investigate the performance of our proposed algorithm with the other two baseline algorithms, namely, *Exhaustive Search* and *Random Search* and determine that our proposed algorithm, *Centroid First*, performs comparable to Exhaustive Search, but with much less computation time.

The remainder of the paper is organized as follows. Section II describes the related works and the basic idea of using MPTCP in SDN network. The definition of the k-partially disjoint paths in a network and the detailed explanation of the proposed algorithm are presented in section III. In section IV, we analyze the performance of our proposed algorithm with the other two methods and section V concludes the paper.

II. BACKGROUND & RELATED WORKS

In this section, we, first, describe some works related to in-band controlling in the SDN and some ideas related to MPTCP. Then, we discuss some works associated with computing the disjoint paths in a network. Finally, we describe the MPTCP based inband controlling that has been introduced in our previous work [4].

For improving the reliability and the high performance of SDN network, Prithviraj *et al*. [7] discuss the problems related to the complexity of SDN controller, the host network, and the data plane. To address these problems along with the deployment of the distributed controller in the SDN network, they present a novel architecture, InitSDN, which designs SDN network in a more flexible and reliable way, without adding any complexity in the controllers. Moreover, to address the problems associated with the scalability, reliability and the uneven load distribution between the distributed controllers, Dixit *et al*. [8] propose ElastiCon, an elastic distributed controller architecture, in which the controller pool expands and shrinks dynamically based on the temporal and spatial variations in traffic conditions and the traffic load is dynamically shifted or moved across different controllers so that each controller can operate within specified work load capacity. Further, they also propose a novel switch migration protocol for enabling the load shifting and balancing across the controllers. In addition, to overcome the overload on the single controller, Hassas Yeganeh and Ganjali [9] propose a novel distributed control plane framework, Kandoo, for scalability of control plane (controllers), which consists of two layers of controllers, first the bottom layer (local controllers), which handles and executes the local control applications that are close to the switches, second, the top layer (centralized root controller), which executes non-local application.

Moreover, Sachin *et al*. [10] propose an in-band control communication mechanism between the switches and the controller by separating the queues for the data and the control traffic. By doing this, control traffic experiences less delay for exchanging the control messages in the network. In addition, failures in the data plane (switch or link failures) can also affect both the data and control traffic in an in-band controlling, so they also discuss the fast failure recovery of the data and control traffic in an OpenFlow based SDN network. In addition, Sharma *et al*. [11] propose an automatic bootstrapping method in which the controller establishes an in-band control channel with other switches. They examine different types of topologies (linear, ring, star and mesh) for investigating the performance of their proposed method. Moreover, for avoiding from the failure situation of both the control and data channels after failure of data plane in the in-band based OpenFlow network, Sachin *et al*. [12] employ two well-known recovery techniques: restoration and protection for in-band based OpenFlow networks, and demonstrate that restoration of the control channel delays in restoration of data channel due to the in-band based network, but the protection of both the channels can achieve the carrier grade recovery requirement.

Multi-path TCP (MPTCP) is the major modification of the regular TCP, transport layer protocol, which uses multiple paths (TCP subflows in MPTCP context) for communication between two devices [13]. The MPTCP connection manages multiple TCP subflows and each TCP subflow works the same as a normal TCP. The multiple subflows may be distributed across multiple disjoint paths between end devices

for the communication. Raiciu *et al*. [6] examine the performance to employ Multi-path TCP in large-scale datacenters. By exploring the multiple paths simultaneously in advance and by linking the congestion response of subflows on these different paths, they determine that traffic can be shifted away from congestion in the network. Furthermore, they demonstrate that MPTCP can be used for both higher utilization of the network and fair allocation of capacity to these subflows of the MPTCP connection in the datacenter networks [6]. They also analyze the performance of single-path TCP and Multi-path TCP with different topologies with different traffic patterns. In addition, Jingpu *et al*. [5] propose a responsive MPTCP system for SDN based datacenters for adjusting the number of subflows for MPTCP connection. Their system has two components: a *centralized controller*, which calculates the additional number of subflow routes intelligently from the current network conditions and a *monitor* (running on each server in the datacenter), which adjusts the number of subflows in datacenter networks. Moreover, due to the popularity and significant benefits of exploiting MPTCP in networks, it has been widely utilized in different wireless networks, so that the users may adopt higher throughput, better user experience, and robust connections using existing wireless technologies (WiFi and 3G/4G) simultaneously in their smart phones [14], [15].

However, if multi-paths are available between a source and destination pair in the network, it may enhance throughput and route resiliency as compared to the single path routing between them. Peter and Sylvie [16] present the performance analysis of multi-path routing and describe how the multi-path routing is suitable for the load balancing mechanism in an ad-hoc network than a single path routing. But multi-paths can contain shared link(s) in their paths and these paths may not be disjoint paths between any source - destination pair. Therefore, it is desirable that a source - destination pair has as many disjoint paths as possible, which can improve the utilization of the bandwidth, lessen the congestion, and decrease the packet drops. Deepinder *et al*. [17] present a distributed distance-vector algorithm that computes multiple disjoint paths having minimum total cost between source and destination pairs.

Moreover, Farabi and Fernando [18] elaborate the Disjoint paths problem, Availability-Based Disjoint paths problem, Maximally Disjoint paths problem, Domain-Disjoint paths problem, Shared Risk Link Group(SPLG) Disjoint paths, and Region Disjoint paths problem in details. Disjoint path pairs problem is also discussed. Zheng *et al*. [19] propose the mathematical models and algorithms that seek the set of K arc-disjoint paths and the set of K vertex-disjoint paths in a network, which can be used alternatively, if link(s) or node fails in a network. Stavros *et al*. [20] propose a solution of network survivability problem by finding the best k-disjoint paths having minimum total cost between a pair of source and destination in Trellis graph technique. Yong and Narasimha *et al*. [21] propose techniques which reduce the backup path computing and finding the disjoint backup path(recovery path) with the primary path and that backup path will be used in case of the fast failure recovery and for the multi-path routing in a network. However, these k-disjoint paths finding algorithms are not well suitable for our proposed model, *MPTCP based in-band controlling in the Software Defined Networking*, described in our previous work because we focus on the disjoint paths from a node (switch) to multiple other nodes (switches), not from a node to another node [4].

For achieving the high performance and low latency in the datacenter, Yu and Deng [22] propose a load balancer for FatTree (well-known topology for the datacenter) with the support of multi-paths in OpenFlow based network. They use dynamic load balancing routing algorithm in their load balancer, which computes multiple flows efficiently by determining the currently available bandwidth on all alternative links and then assigns the paths to each flow in the network for load balancing. Moreover, for the better quality of experience (QoE) of real-time video and the high bandwidth aggregation over multi-paths in the network, Hyunwoo *et al*. [23] propose to dynamically adjust the number of MPTCP subflows (paths) by including and eliminating any MPTCP path from the MPTCP subflow connections (bundle) under varying network conditions and available paths statistics. They also analyze the performance of MPTCP with different flows and SCTP (Stream Control Transmission Protocol) with a single path for downloading the files of different sizes under varying network conditions. In addition, to achieve the efficient and high end-end throughput and better load balancing, Van der Pol *et al*. [24] present prototype which is entirely based on multipath routing using MPTCP in OpenFlow network. Moreover, for avoiding collision and a bottleneck of traffic in OpenFlow network and efficient utilization of multipath routing, Nakasan *et al*. [25] propose a Simple Multipath OpenFlow Controller (SMOC) for splitting and distributing the traffic using MPTCP subflows in the network. SMOC only adopts the current topology information to avoid bottlenecks in the network. SMOC enhances the overall bandwidth utilization of MPTCP connection and network performance. They also compare the performance of SMOC with the POX's original Spanning tree controller. In addition, Marcus *et al*. [26] propose to forward MPTCP subflows over multi-paths so that they can improve the throughput in shared bottlenecks with full utilization of the bandwidth. They can achieve network resiliency by using subflows in different disjoint paths and improve the overall throughput by distributing the different subflows over multi-paths in the network.

Recently, some efforts have been devoted to enhance the resiliency of in-band channels using MPTCP. González *et al*. [27] propose a mechanism which utilizes both the in-band and out-of-band control paths simultaneously for enhancing the OpenFlow control channel via multi-paths using subflows of MPTCP connection. In their proposed solution, they deploy dedicated connection (out-of-band path) from each data plane element to the controller for initializing the OpenFlow control connection and getting

topology related information. As the number of the data plane elements increases, it would be indeed an overhead for the control plane (controller) to manage both the in-band and out-of-band connections. But in our approach, we utilize the data paths across the network for constructing the in-band control channel with the controller through the chosen gate switches. Furthermore, González *et al.* [27] do not discuss well what the advantages of using out-of band paths for the robustness of in-band control connection is and why they obtain topology information through these dedicated data paths. More importantly, the proposed algorithm in [27] only considers a very small topology for computing all available paths between any data plane element and the controller. However, it becomes very complicated to find out all possible available paths between each source and destination data plane element pair in very large size topology. In addition, the paths computed between any pair of source and destination are likely to be partially disjoint paths instead of complete disjoin paths in a large size topology. On the other hand, in our proposed mechanism [4] and proposed algorithms in this work, we consider large size topologies for computing k-partially disjoint paths for in-band control channel between each source node to the controller through the chosen gate switches.

Even though these articles demonstrate some related works on exploiting the SDN network to improve MPTCP performance but they do not employ MPTCP for the improvement and robustness of control channel in SDN network [22], [23], [26]. Therefore, we mainly focus on applying Multi-path TCP (MPTCP) for the control channels in SDN network and our objective here, is to determine the k- partially disjoint paths for the k *gate switches* in SDN networks so that other switches can establish an in-band control channel with the controller through these gate switches by using multiple subflows of the MPTCP connection.

A. MPTCP BASED INBAND CONTROLLING

Since the MPTCP based inband controlling is the principal focus in this paper, we present its comprehensive description in a separate subsection. In the MPTCP based in-band controlling, the controller is connected to a set of k switches of the network, which are designated as the *gate switches*. All other switches have the control connections to the controller through the *gate switches* [4]. Fig. 1 describes an example of the MPTCP based in-band controlling having three ($k = 3$) gate switches, g_1, g_2, and g_3. At the beginning, the controller builds the out-of-band control connections with these *gate switches*. Other switches such as node v establish in-band control connections through these gate switches. Actually, setting up the in-band control connections may determine a step-by-step procedure because the switches near the gate switches can initiate the in-band connections first, and then further away switches can generate the connections later. The detailed procedure for establishing the in-band control connections is out of the scope of this paper and can be discussed in the future work.

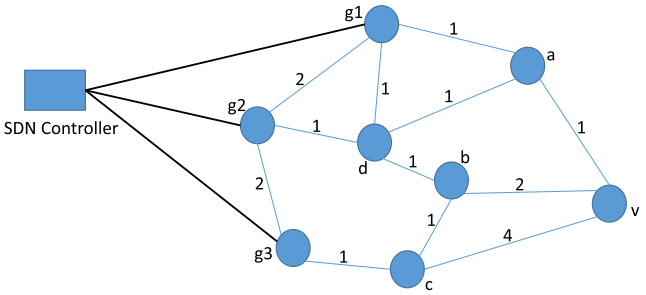


FIGURE 1. An in-band controlling scenario.

In OpenFlow, a control connection is absolutely a TCP/SSL connection between the controller and a switch. However, in the MPTCP based in-band controlling, the control connection is an MPTCP connection instead of a single TCP connection. The number of sub-flows is same as the number of gate switches, k, to maximize the disjointedness of the sub-flows. For example, in Fig. 1, a switch, v, constitutes three MPTCP subflows with the controller through these *gate switches*. Multiple sub-flows in MPTCP enhances the reliability of the control connection in that the breakdown of the data path of a sub-flow does not terminate the control connection. Besides the reliability, such model can also provide load balancing capability because the control messages are distributed over multiple subflow paths while the data traffic in the data plane exists. One thing to perceive is that even though we increase the number of gate switches, k, the disjointedness of the subflows is restricted by the degree of the switch. For example, if the switch has only two adjacent switches, then it can have at most two disjoint paths to two of k gate switches. Thus, we introduce a new definition of disjoint paths to maximize the effectiveness of the large k in the next section.

III. MPTCP BASED IN-BAND CONTROLLING AND GATE SWITCH SELECTION

In this section, we first define the k-partially disjoint paths, which are used for diversifying the subflows for the MPTCP based in-band control connections. Then, we propose an algorithm that computes the k-partially disjoint paths in an efficient way. Finally, we propose a heuristic algorithm that selects the set of k *gate switches* and also explain two baseline algorithms.

A. K- PARTIALLY (EDGE) DISJOINT PATHS

With the fundamental concept of the MPTCP based in-band controlling, in this subsection, we describe the definition of k-partially disjoint paths from a switch to the k gate switches. Actually, it is truly evident that the reliability of the MPTCP based in-band controlling depends on the disjointedness of the sub-flows of the control connections. However, it may not be the straightforward procedure to determine the complete k disjoint paths from a switch to the k gate switches. Furthermore, the disjointedness is limited by the degree of a switch. Thus, in this work, we employ a *loose* definition

of the disjoint paths. When a set of gate switches is elected, the other switches constitute the disjoint paths to the k gate switches based on the definition of the k-partially disjoint paths. For that circumstance, in this subsection, we also introduce a k-partially disjoint path computation algorithm and an enhancement algorithm to reduce the computation time.

- **Definition** a set of k- *partially* edge-disjoint paths: the set of paths in which a path in the set is edge-disjoint with at least one another path in the set.

From now on, we use k- *partially* disjoint paths as the same context as the k- *partially* edge-disjoint paths. For example, in Fig. 1, the three paths, $v-a-g_1$, $v-b-d-g_2$, and $v-b-c-g_3$, are 3-partially disjoint paths. The numbers on the edges are the weights of the edges. It should be noted that these paths to the gate switches may not be the shortest paths to the gate switches. For example, the shortest path from v to g_2 is $v-a-d-g_2$, which holds a shared link with the shortest path $v-a-g_1$. Actually, there can be many such 3-partially disjoint paths. Instead of proposing an algorithm that computes optimal k-partially disjoint paths with some objectives, in this paper, we introduce a simple mechanism for computing a set of k- *partially* edge-disjoint paths from a switch to the k gate switches. Obviously, there are cases where such paths do not exist. However, if there exists one, the proposed algorithm finds it. For the description of the gate switches selection algorithms, we first define some notations.

- Let $G = (V, E)$ be the graph of vertices, V, and edges, E.
- Let $m = |V|$ be the total number of nodes in G, i.e. $V = \{1, 2, 3, \ldots, m\}$.
- Let $w(i, j)$ be the weight of a link (i, j).
- Let k be the number of gate switches in the network.
- Let $\mathcal{P}(k)$ be the set of all possible subsets of k nodes in V.
- Let $S = \{g_1, g_2, \ldots, g_k\} \in \mathcal{P}(k)$ be a set of k gate switches, where $g_j \in V$.
- Let $sp(v, w, G')$ be the shortest path from v to w on a subgraph G' of G.
- Let $sp(v, w)$ be the set of links on the shortest path between a node v and a node w, where $v, w \in V$.
- Let $spl(v, w)$ be the path length of the path $sp(v, w)$.
- Let $dp(v, g_j, S)$ be the set of links on the k-partially *disjoint* path between a node v and a gate switch g_j over the gate switch set S.
- Let $dpl(v, g_j, S)$ be the path length of the path $dp(v, g_j, S)$.

Now, we present a simple algorithm that computes the k-partially disjoint paths from a switch v to k gate switches in $S = \{g_1, g_2, \ldots, g_k\}$. The approximate scheme of the algorithm is as follows. First, we compute the shortest path from v to g_1. Initially, this is considered as the partially disjoint path from v to g_1. Suppose that we have i-partially disjoint paths to the first i gate switches in S. Now, we require to compute a partially disjoint path from v to g_{i+1}. In order to do that, we find the shortest path from v to g_{i+1} in graph G. If that shortest path is edge-disjoint with one of the previously discovered partially disjoint paths up to g_i, this path is the partially disjoint path to g_{i+1}. Otherwise, we remove the edges of the partially disjoint path from v to g_i, which is the *lastly* computed path, then we find the shortest path from v to g_{i+1}. This path is called as the partially edge-disjoint path from v to g_{i+1}. The purpose of removing the *lastly* discovered partially disjoint path is to evenly distribute the paths across the multiple outgoing links of v. By repeating aforementioned procedure up to g_k, the k-partially disjoint paths to S is determined. The proposed algorithm is a *generalization* of the algorithm presented in our previous paper [4], where we consider only the case of $k = 2$. The details of the algorithm are described in Algo. 1.

Algorithm 1 *Disjoint Paths*

1: **Input:** (G, S, v)
2: **Output:** $\{dp(v, g_1), dp(v, g_2), \cdots, dp(v, g_k)\}$
3: S: Set of gate switches, i.e., $\{g_1, g_2, \cdots, g_k\}$
4: v: A Switch
5: Compute shortest paths, $sp(v, g_j)$, from v to each $g_j \in S$
6: $dp(v, g_1) = sp(v, g_1)$ // declare shortest path from v to g_1 as partially edge-disjoint path for g_1
7: $F \leftarrow \{g_1\}$ // save g_1 as first partially edge-disjoint gate switch
8: $S \leftarrow S - F$ // update the gate switch set S by subtracting F
9: **for** $g_j \in S$ **do**
10: **for** $g_l \in F$ **do**
11: flag $\leftarrow$ **false**
12: **if** No shared links in $sp(v, g_j)$ and $sp(v, g_l)$ **then**
13: flag $\leftarrow$ **true** // update flag to *true* if shortest paths for g_j, g_l are edge-disjoint paths
14: **break**
15: **if** flag == **false** // if no partially edge-disjoint gate switches **then**
16: $h \leftarrow$ last added element of F
17: $G' \leftarrow$ remove links on $dp(v, h)$ from G // delete $dp(v, h)$ path from original G
18: $dp(v, g_j) \leftarrow sp(v, g_j, G')$ // Find shortest path for gate switch g_j from updated G'
19: $F \leftarrow F \cup \{g_j\}$ // update partially edge-disjoint gate switch set F by adding g_j
20: $S \leftarrow S - \{g_j\}$ // update the set S by removing gate switch g_j
21:
22: **Return** $\{dp(v, g_1), dp(v, g_2), \cdots, dp(v, g_k)\}$

It should be noted that in line 6 of Algo. 1, we consider the shortest path to g_1, i.e., $sp(v, g_1)$ is also a disjoint path to g_1. We demonstrate this algorithm by illustrating the example in Fig. 1. Suppose we want to compute the 3-partially disjoint paths from node v to gate switches g_1, g_2, and g_3. Initially, the shortest path from v to g_1 ($v-a-g_1$) is the partially disjoint path from v to g_1. Next, we compute the shortest path from v to g_2, which is as follows $v-a-d-g_2$. But this

path is not disjoint with the previously computed partially disjoint path to g_1. Thus we eliminate the edges of the path $v-a-g_1$ and determine the shortest path to g_2 again. The newly computed path is $v-b-d-g_2$, which is considered as the partially disjoint path to g_2. Finally, we compute the shortest path from v to g_3 on G, which is $v-b-c-g_3$. This path is actually disjoint from the partially disjoint path to g_1, namely, $v-a-g_1$. So, we consider this path $v-b-c-g_3$ as the partially disjoint path to g_3. Hence, we successfully determine the 3-partially disjoint paths to the three gate switches.

B. ENHANCEMENT OF DISJOINT PATH COMPUTATION

Actually, we can improve the performance of the Algo. 1 by exploiting natural disjointedness. For example, if the initial k *shortest* paths to k gate switches, $g_j \in S$, are naturally k-partially disjoint paths, we do not need to run Algo. 1. That is the best case. However, even though it is not the best case, some pairs of the k *shortest* paths might be disjoint. In such situation, we expect to employ Algo. 1 only for those g_js that do not produce disjoint shortest paths. To be precise, in Algo. 1, in line 7, we introduce F as the set of the gate switches that provide the natural disjoint shortest paths. The rest of the algorithm is identical. We characterize this procedure as the *Pre-Checking* operation for the k-partially disjoint path finding algorithm. The *Pre-Checking* algorithm requires only $O(k^2)$ number of path comparison operation, in which we only necessitate to investigate whether a pair of shortest paths are the edge-disjoint or not.

However, this Pre-Checking method incorporates a very interesting property, which can be utilized to lessen the Pre-Checking computation time even further. Essentially, the shortest paths are usually k-partially disjoint paths and if not, no shortest path finds another the shortest path that is disjoint. In other words, sometimes the shortest paths are actually k-partially disjoint paths. Therefore, we can consider the shortest paths as k-partially disjoint paths. Otherwise, we need to run the Algo. 1 with $F \leftarrow \{g_1\}$. There is no intermediate case left where F holds a proper subset of k gate switches. The proof of this property is given in Theorem 1.

Theorem 1 (Natural Disjointedness): Among the shortest paths from a switch to the gate switchs, if a shortest path is disjoint with another shortest path, the shortest paths are actually k-partially disjoint paths.

Proof of Theorem 1: Suppose that there are k shortest paths from a switch v to k gate switches $\{g_1, g_2, \cdots, g_k\}$ and that the shortest paths v to g_i ($v \rightarrow g_i$) and v to g_j ($v \rightarrow g_j$) are actually disjoint. We claim that all the other paths are disjoint with either $v \rightarrow g_i$ or $v \rightarrow g_j$. If so, all the shortest paths have at least one disjoint shortest path.

Let a link (x, y) be the *last* link on the shortest path $v \rightarrow g_l$ that is shared by either $v \rightarrow g_i$ path or $v \rightarrow g_j$ path (Refer to Fig. 2). Since $v \rightarrow g_i$ and $v \rightarrow g_j$ are disjoint, (x, y) belongs to only one of the two shortest paths. Suppose that (x, y) is shared with $v \rightarrow g_i$ without loss of generality. Then the $y \rightarrow g_l$ segment of the shortest path $v \rightarrow g_l$ is disjoint from the shortest path $v \rightarrow g_j$ because (x, y) is the last link

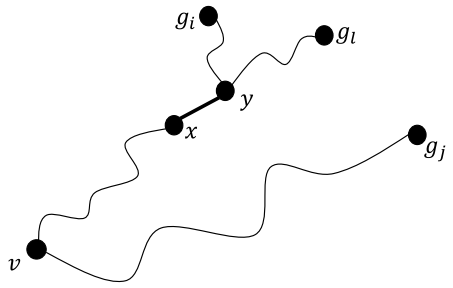


FIGURE 2. An example for the Proof of Theorem 1.

on the shortest path $v \rightarrow g_l$ that is shared by either $v \rightarrow g_i$ path or $v \rightarrow g_j$ path. Furthermore, the $v \rightarrow y$ segment of the shortest path $v \rightarrow g_l$ should be the same as the $v \rightarrow y$ segment of the shortest path $v \rightarrow g_i$ because the paths $v \rightarrow g_i$ and $v \rightarrow g_l$ are *shortest* paths due to the optimality of the shortest paths. Thus, $v \rightarrow y$ segment of $v \rightarrow g_l$ is disjoint from $v \rightarrow g_j$. Since both $v \rightarrow y$ and $y \rightarrow g_l$ segments of $v \rightarrow g_l$ are disjoint from $v \rightarrow g_j$, the shortest path $v \rightarrow g_l$ is disjoint from the shortest path $v \rightarrow g_j$. □

Theorem 1 presents a useful way to reduce the computation time to examine whether the shortest paths are actually k-partially disjoint paths. The Pre-checking procedure works as follows. First, we compute the shortest paths to the k gate switches. Then, we take one shortest path and check whether this shortest path is disjoint with any other shortest path. If do so, we comprehend that the shortest paths are actually the k-partially disjoint paths. This procedure adopts only $O(k)$ path comparison operations whether or not the shortest paths are actually k-partially disjoint path.

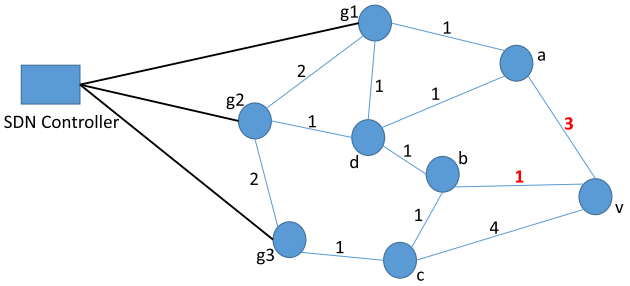


FIGURE 3. An in-band controlling scenario.

Fig. 3 exhibits such an example, which is identical to the Fig. 1 with the weight changes of two links: $(v-a)$ and $(v-b)$. The shortest paths from v to gate switches (g_1, g_2, g_3) are $v-b-d-g_1$, $v-b-d-g_2$, and $v-b-c-g_3$, respectively. It should be noted that these three shortest paths are not actually disjoint between any pair of the shortest paths. If we employ the *Pre-Checking* property that we discussed earlier, we first take the path $v-b-d-g_1$ and compare it with $v-b-d-g_2$. The two paths are not actually disjoint. Then we compare the shortest path, $v-b-d-g_1$ with another shortest path $v-b-c-g_3$, which are not disjoint, either. Thus, we can conclude that the shortest paths of node v to the gate

switches are not k-partially disjoint paths. Let's take another example with the switch c. The shortest paths from the switch c to the three gate switches are $c-b-d-g_1$, $c-b-d-g_2$, and $c-g_3$, respectively. Suppose we take the shortest path, $c-g_3$ and compare it with the other shortest paths. If we compare $c-g_3$ with the shortest path, $c-b-d-g_1$, they are actually disjoint paths. Then, we can immediately conclude that the three shortest paths from a switch c to the gate switches are k-partially disjoint paths based on the Theorem 1.

C. GATE SWITCH SET SELECTION METHODS

We have described the definition of k-partially disjoint paths so far. In the MPTCP based in-band controlling, the set of gate switches of length k is selected first, and then, other switches utilize the k-partially disjoint paths determined by the algorithm described in the previous subsection. So, the performance of the in-band controlling extremely depends on how to select the gate switches. For that matter, we need to define the objective of selecting the gate switches. Roughly speaking, we want to select gate switches in such a way that the disjoint path lengths from a switch to the chosen gate switches are as small as possible. By performing this way, we can reduce the latency of the control messages that are exchanged between a switch to the controller. Determining such paths is a *challenging problem* [4]. We have suggested several gate switch selection methods only for $k = 2$ in our previous work [4]. However, we observe that the methods proposed in our previous work are designed inefficiently [4]. Thus, in this paper, we propose a heuristic for gate switch selection method for an arbitrary k, the size of gate switch set.

Before we present the heuristic algorithm, we describe the objective function for the gate switch selection problem. In brief explanation, we want to compute a *gate switch* set, S, of size k that satisfies the objective function (1).

$$\underset{S\in\mathcal{P}(k)}{\text{minimize}} \quad \sum_{v\in V}\sum_{g_j\in S} dpl(v, g_j, S) \tag{1}$$

Basically, for the given gate switch set S, we compute the sum of k-partially disjoint path lengths from all the switches to all the gate switches. By achieving (1), we can have shorter k-partially disjoint paths. However, as we mentioned earlier, achieving (1) is considerably challenging. We conjecture that this is an NP-hard problem. So we propose a heuristic algorithm.

Before we propose the heuristic algorithm, we describe two baseline algorithms: *Exhaustive Search* and *Random Search*. In *Exhaustive Search*, we examine every possible set S, $S \in \mathcal{P}(k)$. For each $S \in \mathcal{P}(k)$, we employ the k-partially disjoint path finding algorithm, Algo. 1. Then we measure the sum of disjoint path lengths for S. Then, finally, we choose set S that achieves (1). The *Exhaustive Search* method is described in Algo. 2.

However, the *Exhaustive Search* demands to check $\binom{|V|}{k}$ gate switch sets and for each test, approximately k times shortest path computing is required, which leads to an intolerably long time.

Algorithm 2 *Exhaustive Search*

Input: G, k
Output: S^*
$md \leftarrow \infty$ // set maximum distance md to ∞
$S^* \leftarrow \phi$ // declare best gate switch set to empty set initially
for each S in $\mathcal{P}(k)$ **do**
  $d \leftarrow 0$ // set total distance d to 0 for each set S
  for each $v \in V$ **do**
    Find the k-partially edge disjoint paths from i to S
    for each $g_j \in S$ **do**
      $d \leftarrow d + dpl(v, g_j)$ // update d by adding dpl of each g_j in set S
  // if distance d of current set S is less than maximum distance md
  if $d < md$ **then**
    $md \leftarrow d$ // update maximum distance md by d
    $S^* \leftarrow S$ // update best set S^* by switches of set S

Return S^*

Algorithm 3 *Random Search*

Input: G, k, r
Output: $\hat{S}$
r is the number of iterations
$md \leftarrow \infty$ // set maximum distance md to ∞
$\hat{S} \leftarrow \phi$ // declare best gate switch set to empty set initially
for $cnt = 0, cnt{+}{+}$, while $cnt < r$ **do**
  $S \leftarrow$ a randomly selected switch set from $\mathcal{P}(k)$
  $d \leftarrow 0$ // set total distance d to 0 for each set S
  for each $v \in V$ **do**
    Find the k-partially edge disjoint paths from i to S
    for each $g_j \in S$ **do**
      $d \leftarrow d + dpl(v, g_j)$ // update d by adding dpl of each g_j in set S
  // if distance d of current set S is less than maximum distance md
  if $d < md$ **then**
    $md \leftarrow d$ // update maximum distance md by d
    $\hat{S} \leftarrow S$ // update best set $\hat{S}$ by switches of set S

Return $\hat{S}$

Next, we illustrate another baseline algorithm called the *Random Search*, which is described in the Algo. 3. Basically, unlike the Exhaustive search, instead of examining all sets of k switches in $\mathcal{P}(k)$, we randomly choose a small number of elements in $\mathcal{P}(k)$ and for each selected element, we exploit the k-partially disjoint path finding algorithm, Algo. 1. The performance of Random Search totally depends on the number of switch sets that we examine. If the number is large, we have a high chance to discover a better solution while having the significantly long computation time, and vice versa. As it

can be seen in Algo. 3, the only difference between Random Search and Exhaustive Search algorithms is the number of elements in $\mathcal{P}(k)$ to be examined.

Now, we propose a heuristic algorithm called *Centroid First*. This algorithm is based on the following heuristic. Intuitively, when a gate switch set S shows a small sum of *disjoint* path lengths, S is likely to show a small sum of *shortest* path lengths. Even though the reverse is not always true, it is likely that when a gate switch set S reveals a small sum of the *shortest* path lengths, S might give a small sum of *disjoint* path lengths. Our heuristic algorithm exploits this intuition. Consequently, we obtain the top k switches that have the least total sum of shortest path lengths from themselves to all the other switches. We just consider the discovered k switches as the gate switches. Then, we compute the k-partially edge-disjoint paths from the switches to the chosen k gate switches. Since the top k switches can be thought of the centroids of the network, we name the heuristic algorithm *Centroid First*. The detailed description of the *Centroid First* algorithm is presented in Algo. 4.

Algorithm 4 *Centroid First*

1: **Input:** G, k
2: **Output:** $\hat{S}$
3: **for** each v in V **do**
4: $pl(v) \leftarrow 0$ // set path length pl of node v to 0
5: **for** each $w \in V$ **do**
6: $pl(v) \leftarrow pl(v) + spl(v, w)$ // update path length of node v by adding spl between v and w
7:
8: **Return** $\hat{S} = \{v: v \in V$ which have k smallest $pl(v)\}$

As for the illustration of our proposed algorithm, if we employ the *Centroid First* to the topology in Fig. 1, the selected switches would be b and d for $k = 2$ case because the two nodes have the smallest total sum of path lengths. As can be observed in this example, the gate switches by the Centroid First algorithm are destined to be in the center of the networks. Those switches are likely to have a small sum of path lengths.

IV. PERFORMANCE EVALUATION

In this section, we investigate the performance of our proposed, *Centroid First* algorithm, along the with the two baseline algorithms: the *Random search* and the *Exhaustive search*. For the evaluation, we generate random topologies by using BRITE topology generator [28]. The node sizes of the random topologies are 100, 200, 300, and 400. The average degrees of the topologies are 4 and 6. For each node size and degree pair, we randomly generate 10 different topologies. Thus, the total number of randomly generated topologies is 80. The three algorithms are implemented with C++ language and executed on Ubuntu virtual machine running on VirtualBox environment. We adopt two performance evaluation metrics: *average disjoint path length* and *average computation time (s)*. It should be noted that the evaluation results are the averages from the 10 random topologies of each size and degree.

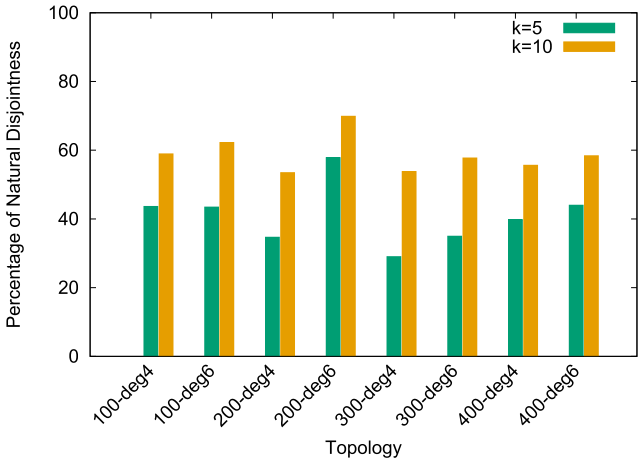


FIGURE 4. Percentage of natural k-partially disjoint paths over various topologies, $k = 5,10$.

Before we examine the performance of *Centroid First* algorithm, we analyze the natural disjointness of the topologies. Fig. 4 indicates the percentage for 8 different topology types with $k = 5, 10$ as the gate switch set, selected by the Centroid First algorithm. There are around 40% of natural k-partially disjoint paths for $k = 5$ and around 50% for $k = 10$. Moreover, the trends are similar for both k values for different topologies. So if we have large k, then regardless of the number of nodes, there is a high possibility to have natural disjoint paths.

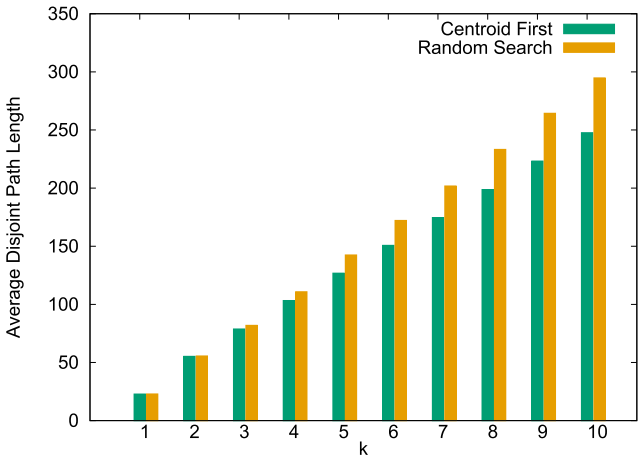


FIGURE 5. Average disjoint path length for 400 node degree 4 topologies over k, 1000 iterations for Random Search.

Since the average disjoint path lengths of the Random search and the Centroid First vary for the different k, we now vary the value of k from 1 to 10 and examine the variation of average disjoint path lengths. We compare the results for the Centroid First and Random search. We are not able to produce the results of Exhaustive search since it is tremendously slow to complete. However, we concentrate here to deeply investigate the performance of Random Search when k varies. Fig. 5 exhibits the average disjoint path lengths

over k for 400 node degree 4 topologies. As can be seen in Fig. 5, the Centroid First algorithm shows the smaller average disjoint path lengths than Random Search except for $k = 1$ and 2. It means that when k is small, the number of all possible sets of k gate switches is also small, i.e., $|\mathcal{P}(k)|$ is small. Thus, 1000 samples may contain good gate switch sets. However, when k is large, 1000 samples may not be enough. The difference between the average disjoint path lengths of the Centroid First and Random search becomes larger when the value of k becomes larger. We observe the same trend for other types of topologies. This result is quite intuitive because when the sample space set or population set is large, the Random search usually does not perform well.

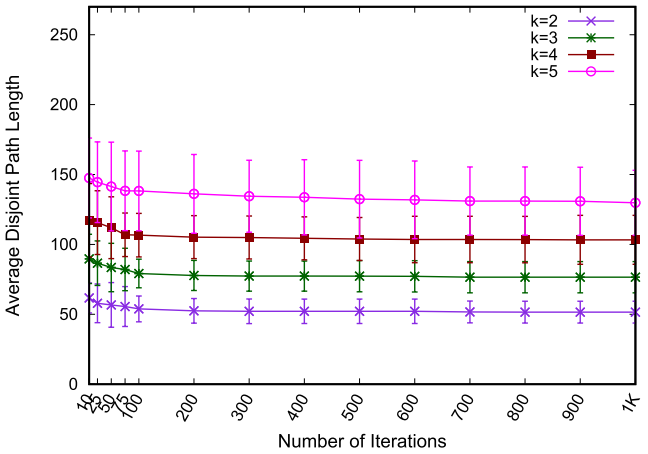


FIGURE 6. Average disjoint path length over the number of iterations for 200 node degree 4 topologies, $k = 2,3,4,5$.

We investigate the performance of Random search further to check whether the number of iterations influences the results much. We vary the number of iterations from 10 to 1000 with various k values for the 200 node degree 4 topologies. Fig. 6 represents the average disjoint path lengths over the number of iterations altering from 10 to 1000. The average disjoint path lengths become constant when the number of iterations increases, especially over 500 iteration value. Fig. 6 also indicates the minimum and maximum average disjoint path lengths. The ranges also become stable when the number of iterations approaches to 1000. Especially for $k = 5$, the difference of the average disjoint path length is small for the different number of iterations. This shows that expanding the number of iterations does not improve the average disjoint path lengths a lot unless the number of iterations becomes comparable to the population size. We experience the similar trend for other types of topologies with different number of nodes and degrees.

Now, we investigate the average disjoint path lengths of the three algorithms for 8 different types of topologies, with "$k = 2$ and 3" as the gate switch set sizes. For the Random approach, we adopt 1000 iterations, i.e., the number of randomly selected gate switch sets is 1000. We run each algorithm and compute the average disjoint path lengths.

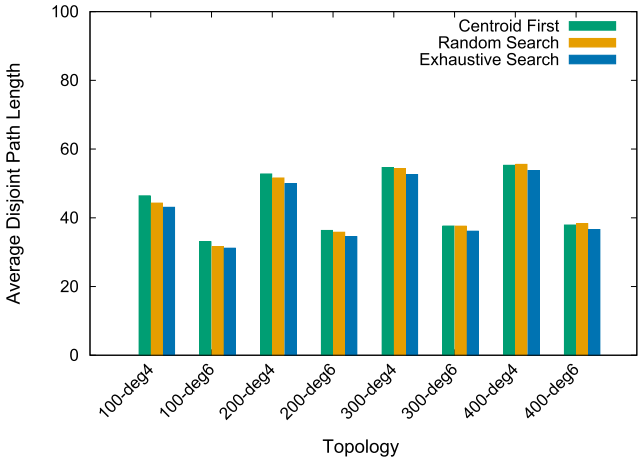


FIGURE 7. Average disjoint paths length for various topologies, $k = 2$.

Fig. 7 presents the average disjoint path lengths when $k = 2$. Fig. 8 shows the results when $k = 3$ as the number of gate switches. The labels in the x-axis represent the number of nodes and the average degree of the nodes. For example, 100-deg4 indicates a topology of 100 nodes with the average degree of 4.

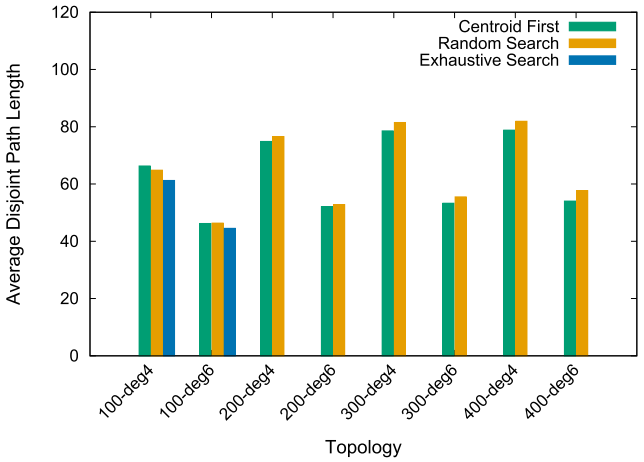


FIGURE 8. Average disjoint paths length for various topologies, $k = 3$.

As can be seen in Fig. 7 and Fig. 8, all the three algorithms present similar average disjoint path lengths. However, when the topology size increases, the Centroid First performs better than the Random Search. It should be noted that the computation time of Exhaustive search is very large for larger topologies and large value of k, so we have missing results of the Exhaustive search for $k = 3$.

So far, we have investigated the performance in terms of the average disjoint path lengths. The performances of the three algorithms are actually comparable except that the Random performs the worse when the topology size increases. Now, we investigate the computation time of the three algorithms. Fig. 9 represents the computation time in seconds over various k values for the 200 node degree 4 topologies.

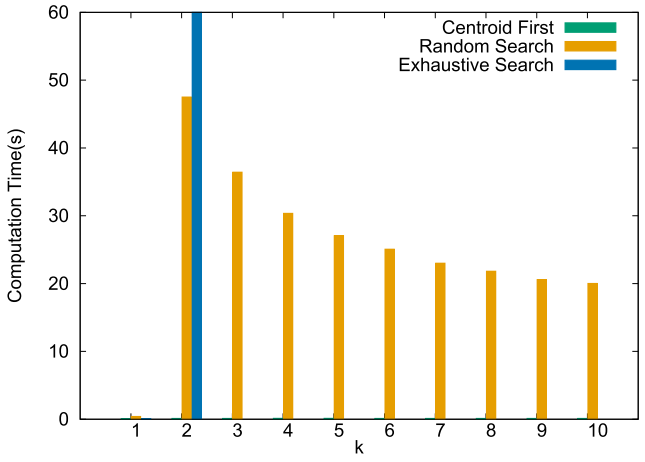


FIGURE 9. Computation time over k, 200 node degree 4 topologies.

The computation time is the average of 10 computation times to complete the simulation for the 10 different topologies of same size and degree. We just display the result of one type of topologies because the results of other types of topologies are quite similar. As it can be observed in the Fig. 9, the computation times of Random search and Exhaustive search are extremely high. Especially, that of the Exhaustive search is very high so that the simulations for large k values cannot be finished. One interesting thing is that the computation time of Random search declines when k increases. This is because when k is large it requires less time to determine the k-partially disjoint paths from the switches to the selected gate switches by exploiting the optimization of Pre-Checking presented in the previous section. In other words, the shortest paths are likely to be k-partially disjoint paths when k is high. The computation time of the Centroid First is quite small so it is difficult to be observed in the figure.

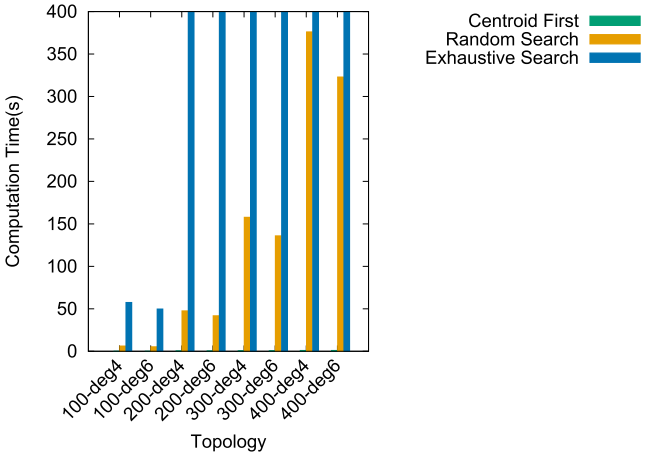


FIGURE 10. Computation time (s) for various topologies, $k = 2$.

To further investigate the computation time, we vary the types of topologies for $k = 2$ case. Fig. 10 represents the *computation time* in seconds of the *three* algorithms in case of 100, 200, 300 and 400 nodes with degree 4 and 6 topologies with $k = 2$. The computation time of Random search increases as the number of nodes increases. This is actually expected because the k - partially disjoint path computation time rises when the number of nodes increases. One obvious fact is that the Exhaustive search takes considerably high computation time because the number of all possible subset of k nodes, $|\mathcal{P}(k)|$, increases. Moreover, the Centroid First algorithm experiences very less computation time, which are displayed by small ticks for larger sizes topologies in Fig. 10.

In summary, the *Centroid First* algorithm exhibits smaller *Average disjoint path lengths* than *Random Search* method for various type of topologies and even for the higher values of k and also presents comparable *Average disjoint path lengths* to the *Exhaustive Search*. Furthermore, *Centroid First* experiences very small *computation time* unlike the other two algorithms: *Random Search* and *Exhaustive Search*. The extensive simulation results clearly indicate that *Centroid First* is a viable option for selecting the k gate switches for the in-band controlling in SDN.

V. CONCLUSION

MPTCP based In-band controlling in SDN uses data paths for the control connection. Thus, it requires to select good gate switches for the connection to be robust. In this paper, we address the gate switch selection problem, where a set of k gate switches is selected in such a way that the average disjoint path lengths become small. To ensure the better availability and reliability of MPTCP based in-band control channel between the switches with the controller, we propose *Centroid First* algorithm, which selects a set of k *gate switches* by exploiting the intuition that the set of switches that provide smaller *shortest* path lengths from other switches may provide small *disjoint* path lengths.

Through extensive simulations, we evaluate whether the proposed *Centroid First* performs well, thus confirms our intuition. We compare the performance of *Centroid First* with two baseline algorithms: the *Random Search* and the *Exhaustive Search*. The "Centroid First" algorithm performs the best with even for the higher values of k than the *Random Search* method and additionally, it has very low *computation time* as compare to that of both the *Random Search* and the *Exhaustive Search* methods. In the future, we plan to implement MPTCP based in-band controlling mechanism to analyze the performance in the real SDN environment. Furthermore, the detailed procedure of constructing the in-band control connections will be discussed in the future work. If the in-band control connections are constructed in a plug-and-play way, it will ease the work of the network operators.

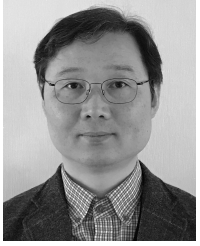
REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[3] (Sep. 6, 2012). *OpenFlow Switch Specification, Version 1.3.1 (Wired Protocol 0x04)*. Accessed: Nov. 30, 2018. [Online]. Available: https://www.opennetworking.org/sdn-resources/openflow
[4] A. Raza, A. Gohar, and S. Lee, "MPTCP based in-band controlling for the software defined networks," in *Proc. Int. Conf. Inf. Commun. Technol. Converg.*, Jeju, South Korea, Oct. 2017, pp. 163–167.
[5] J. Duan, Z. Wang, and C. Wu, "Responsive multipath TCP in SDN-based datacenters," in *Proc. IEEE Int. Conf. Commun.*, London, U.K., Jun. 2015, pp. 5296–5301.
[6] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proc. ACM SIGCOMM Conf.*, Toronto, ON, Canada, 2011, pp. 266–277.
[7] P. Patil, A. Gokhale, and A. Hakiri, "Bootstrapping software defined network for flexible and dynamic control plane management," in *Proc. 1st IEEE Conf. Netw. Softwarization*, London, U.K., Apr. 2015, pp. 1–5.
[8] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw.*, Hong Kong, 2013, pp. 7–12.
[9] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Helsinki, Finland, 2012, pp. 19–24.
[10] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "In-band control, queuing, and failure recovery functionalities for OpenFlow," *IEEE Netw.*, vol. 30, no. 1, pp. 106–112, Jan./Feb. 2016.
[11] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Automatic bootstrapping of OpenFlow networks," in *Proc. 19th IEEE Workshop Local Metrop. Area Netw.*, Brussels, Belgium, Apr. 2013, pp. 1–6.
[12] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Fast failure recovery for in-band OpenFlow networks," in *Proc. 9th Int. Conf. Design Reliable Commun. Netw. (DRCN)*, Budapest, Hungary, Mar. 2013, pp. 52–59.
[13] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *MPTCP: TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 6824, Internet Requests for Comments, Internet Engineering Task Force, Fremont, CA, USA, Jan. 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824
[14] Y.-C. Chen, Y.-S. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath TCP performance over wireless networks," in *Proc. Internet Meas. Conf.*, Barcelona, Spain, 2013, pp. 455–468.
[15] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure, "Exploring mobile/WiFi handover with multipath TCP," in *Proc. ACM SIGCOMM Workshop Cellular Netw., Oper., Challenges, Future Design*, Helsinki, Finland, 2012, pp. 31–36.
[16] P. P. Pham and S. Perreau, "Performance analysis of reactive shortest path and multipath routing mechanism with load balance," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun.*, San Francisco, CA, USA, Apr. 2003, pp. 251–259.
[17] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in *Proc. Commun. Archit. Protocols Conf.*, Zurich, Switzerland, Sep. 1991, pp. 43–51.
[18] I. Farabi and A. Fernando, "Disjoint paths in networks," in *Wiley Encyclopedia of Electrical and Electronics Engineering*. Hoboken, NJ, USA: Wiley, Jun. 2015.
[19] Z. Xie, H. Leng, Z. Chen, and J. Zhang, "Finding arc and vertex-disjoint paths in networks," in *Proc. 8th IEEE Int. Conf. Dependable, Autonomic Secure Comput.*, Chengdu, China, Dec. 2009, pp. 539–544.
[20] S. D. Nikolopoulos, A. Pitsillides, and D. Tipper, "Addressing network survivability issues by finding the k-best paths through a trellis graph," in *Proc. 16th Annu. Joint Conf. IEEE Comput. Commun. Soc.*, Kobe, Japan, Apr. 1997, pp. 370–377.
[21] Y. O. Lee and A. L. N. Reddy, "Constructing disjoint paths for failure recovery and multipath routing," *Comput. Netw.*, vol. 56, no. 2, pp. 719–730, 2012.
[22] L. Yu and P. Deng, "Openflow based load balancing for fat-tree networks with multipath support," in *Proc. 12th IEEE Int. Conf. Commun.*, Budapest, Hungary, Jun. 2013, pp. 1–5.
[23] N. Hyunwoo, C. Doru, and H. Schulzrinne, "Towards dynamic MPTCP path control using SDN," in *Proc. 2nd IEEE Conf. Netw. Softwarization*, Seoul, South Korea, Jun. 2016, pp. 286–294.
[24] R. van der Pol *et al.*, "Multipathing with MPTCP and OpenFlow," in *Proc. SC Companion, High Perform. Comput., Netw. Storage Anal.*, Salt Lake City, UT, USA, Nov. 2012, pp. 1617–1624.
[25] C. Nakasan, K. Ichikawa, H. Iida, and P. Uthayopas, "A simple multipath OpenFlow controller using topology-based algorithm for multipath TCP," *Concurrency Comput., Pract. Exper.*, vol. 29, p. e4134, Mar. 2017.
[26] M. Sandri, A. Silva, L. A. Rocha, and F. L. Verdi, "On the benefits of using multipath TCP and OpenFlow in shared bottlenecks," in *Proc. 29th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Gwangiu, South Korea, Mar. 2015, pp. 9–16.
[27] S. González, A. de la Oliva, C. J. Bernardos, and L. M. Contreras, "Towards a resilient OpenFlow channel through MPTCP," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Valencia, Spain, Jun. 2018, pp. 1–5.
[28] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal topology generation from a user's perspective," Boston Univ., Boston, MA, USA, Tech. Rep. BU-CS-TR-2001-003, May 2001.

ALI RAZA received the B.S. degree in computer engineering from the University of Engineering and Technology, Taxila, Rawalpindi, Pakistan. He is currently pursuing the master's degree with the School of Computer Science, Kookmin University, Seoul, South Korea. His research interests include the different fields of communication networks and its algorithms, software defined networks, and data center networks technologies.



SANGHWAN LEE received the B.S. and M.S. degrees from Seoul National University, South Korea, in 1993 and 1995, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering, University of Minnesota, in 2005. After receiving the M.S. degree, he was with Hyundai Electronics, South Korea, for five years. From 2005 to 2006, he was with the IBM Thomas J. Watson Research Center. He then joined Kookmin University, Seoul, South Korea, in 2006. His main research area is in the fields of software defined networks, scalable routing selection for multimedia, and different theory and services needed in Internet.