Received September 3, 2019, accepted September 24, 2019, date of publication October 11, 2019, date of current version October 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2946707

Comparison of Routing Algorithms With Static and Dynamic Link Cost in Software Defined Networking (SDN)

ERDAL AKIN AND TURGAY KORKMAZ
The University of Texas at San Antonio, San Antonio, TX 78249, USA
Corresponding author: Erdal Akin (erdal.akin@utsa.edu)

ABSTRACT In this paper, we compared the existing routing algorithms in the context of Software Defined Networking (*SDN*), where a logically centralized controller acquires the global view of the network, selects the paths using a routing algorithm, and installs the determined routing rules to the switches. We divided existing routing algorithms (RA) into three categories: RA with static link cost (*RA-SLC*), RA with dynamic link cost (*RA-DLC*), and RA with dynamic link cost and minimum interference (*RA-DLCMI*). We then implemented various routing algorithms from each category using *RYU SDN* controller and compared their performances on *Mininet* emulator. For the network state information (NSI) needed for the routing algorithms, we first consider the idealistic case where the accurate NSI is available, as assumed in the literature. However, since this is not possible in practice, we also considered the practical case where the controller periodically collects the NSI with some inaccuracy. In our experiments, we observed that while *RA-DLC* and *RA-DLCMI* give similar performance, *RA-SLC* falls behind of *RA-DLC* and *RA-DLCMI* in the number of accepted flows and total throughput. We also showed that the performance of every algorithm is adversely affected from the inaccuracy of NSI, calling for further research on developing effective NSI collection methods.

INDEX TERMS Routing algorithms, network state information accuracy, software defined networking (SDN), traffic engineering.

I. INTRODUCTION

Traditional IP Network was originally designed as a simple but highly fault-tolerant distributed system that can provide best-effort communications services in a trusted environment. However, these key assumptions behind the original design have been changing in the real-world situations, requiring the Internet to have many new features. Unfortunately, it was almost impossible to replace or redesign the underlying IP architecture [1]–[3]. As a result, many new features were simply patched on top of the original architecture, resulting in one of the largest and complex distributed system to manage.

To simplify network management while creating a flexible architecture that can seamlessly integrate new features into the network, researchers have introduced a recent model called Software-Defined Networking *(SDN)* [4], [5]. In essence, SDN separates the *control plane* from *data plane*, and builds up a logically centralized controller. This logically centralized controller is responsible for obtaining the global view of the underlying network and making all control plane decisions (e.g., which paths to use) [6]–[8]. The controller then simply conveys these decisions to the underlying routers and switches via a well-defined *API* such as *OpenFlow* [9]. This centralized control plane greatly simplifies network configuration and new policy installation, particularly when compared to the traditional distributed control plane where each router/switch has to figure out what to do through exchanging massages with each other.

Realizing the potential benefits of SDN, the industry has also shown significant interest in developing and deploying SDN-based technologies. For example, many commercial switches/routers have now support for *OpenFlow API* [6]. An *OpenFlow switch* is simply a forwarding element (such as a router, switch, firewall, etc.) that forwards incoming packets based on the routing decisions that are determined and installed by SDN controllers [1], [6], [7].

Efficiently *routing* the given flows through the underlying network is one of the important objections in SDN. To achieve this, the centralized controller needs to efficiently perform the following three tasks: (*i*) acquiring the global view of

The associate editor coordinating the review of this manuscript and approving it for publication was Zhaojun Li.

the underlying network as accurately as possible, (*ii*) computing the best paths that can meet the demands of the given flows while highly utilizing the underlying resources, and (*iii*) installing the determined routing rules to the forwarding elements (e.g., switches and/or routers) in data plane. Since an open communication protocol (e.g., OpenFlow [7]) is used in order to operate the last task, we especially concentrate on the first two tasks. Accordingly, we implemented several existing routing algorithms and compared their performance. For the implementation and testing, we used *RYU* controller [10] and *Mininet* emulator [11]. We performed our experiments under two different network topologies while obtaining network state information using both the idealistic case where we maintain accurate NSI in the controller and practical case where the controller periodically collects NSI and has some inaccuracies.

Various surveys have focused on Quality of Service (QoS) routing algorithms under different network applications and settings. For example, surveys related to multi-cast routing algorithms that aim at finding paths from one source to multiple destinations have been studied in [12]–[15]. Multi-path routing algorithms have been studied in [16]. Several other surveys have focused on routing algorithms in wireless networks [17]–[22]. A recent survey in [23] has provided a detailed description of various QoS routing algorithms while complementing several prior surveys focusing on general QoS routing and its challenges [24]–[26]. The authors in [23] have especially focused on delay constrained routing algorithms and compared their performance. In contrast, we mainly focus on QoS routing algorithms that use available bandwidth (or utilization) as cost metrics. All the above mentioned surveys have been assuming that accurate network state information (NSI) is available. However, collecting NSI is not an easy task as we have presented in [27]. Accordingly, in this paper, we also considered the periodic NSI collection method with different intervals and evaluate how the routing algorithms perform under different level of inaccuracies in the acquired NSI.

The existing routing algorithms use either *static link cost* (e.g., hop count, distance, link capacity) or *dynamic link cost* (e.g., available link capacity, link utilization). To further improve the routing algorithms using dynamic link cost, researchers have considered additional metrics (e.g., number of flows on a link) to *minimize the interference* among the flows. Correspondingly, we grouped the existing routing algorithms under three classes and especially implemented the followings:

- RA-SLC: Routing Algorithms with *Static Link Cost* include Minimum Hop Algorithm (*MHA*), Shortest Path Algorithm (*SP*) and Widest Shortest Path Algorithm (*WSP*) [28]–[30].
- RA-DLC: Routing Algorithms with *Dynamic Link Cost* include Constraint Shortest Path First (i.e., Dynamic Shortest Path (*DSP*)), Dynamic Widest-Shortest Path Algorithm (*DWSP*).
- RA-DLCMI: Routing Algorithms with *Dynamic Link Cost and Minimum Interference* include Minimum Interference Routing Algorithm (*MIRA*), the Least Interference Optimization Algorithm (*LIOA*) and the Improved Least Interference Routing Algorithm (*ILIOA*) [30]–[33].

All of the above mentioned routing algorithms have been assuming that the accurate network state information (NSI) is available in the controller. In the case of a single controller, we will try to mimic this assumption by updating the NSI at the controller as we install new flows, which we call a shadow topology. However, in practice, the controllers have to query NSI from the switches because of the background traffic and multiple controllers that may install rules for different flows. As we discuss later, obtaining accurate NSI in such practical scenarios is a challenging issue [27], [34], [35] and negatively affects the performance of every routing algorithm, calling for further research.

In our experiments, *RA-DLC* and *RA-DLCMI* outperform *RA-SLC* as we expected. We also observed that the existing algorithms using dynamic link cost do not have significant difference between each other under both accurate NSI and periodic NSI. However, since periodically collected NSI is not accurate, more flows have been accepted by all algorithms which caused overload on the network, resulting in more packet loss for every flow. Obviously, even though total throughput is high, packet loss is not acceptable for the sake of Quality of Service (*QoS*). In Section V, we will discuss more results in detail.

The remainder of this paper is structured as follows. We first give detailed background information related to *SDN* in Section II. In Section III, we describe existing routing algorithms and give their computational complexities. In Section IV, we discuss how we mimic the availability of accurate NSI in the controller and how NSI is periodically collected in practice. In Section V, we explain our experimentation set up and discuss our results. Finally, we present our conclusions and some directions for future research in Section VI.

II. BACKGROUND OF SOFTWARE DEFINED NETWORKING

A. WHAT IS SOFTWARE DEFINED NETWORKING?

Software-Defined Networking is a new network architecture where the network's control logic (the Control Plane) and forwarding logic (the Data Plane) are decoupled [1], [36]. After this separation, the central controller becomes the brain of the network and makes all the decisions for the routers and switches that become simple forwarding devices. As illustrated in Figure 1, this is in sharp contrast to the traditional IP network where the control logic and forwarding logic are coupled and distributed [37].

The main components and the general architecture of SDN are shown in Figure 2. From this figure, we can see that SDN

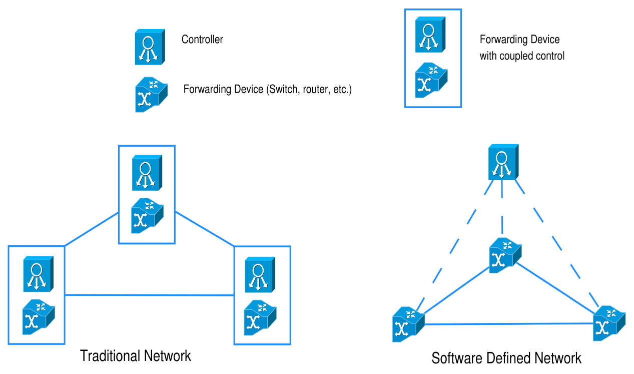

Controller
Forwarding Device (Switch, router, etc.)
Forwarding Device with coupled control
Traditional Network
Software Defined Network

FIGURE 1. Comparing traditional network and SDN.

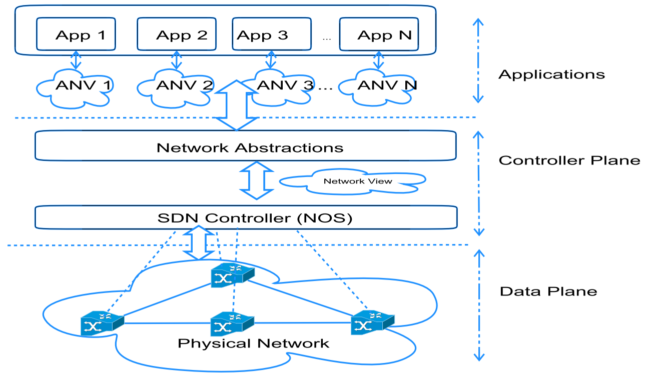


FIGURE 2. Software defined network architecture.

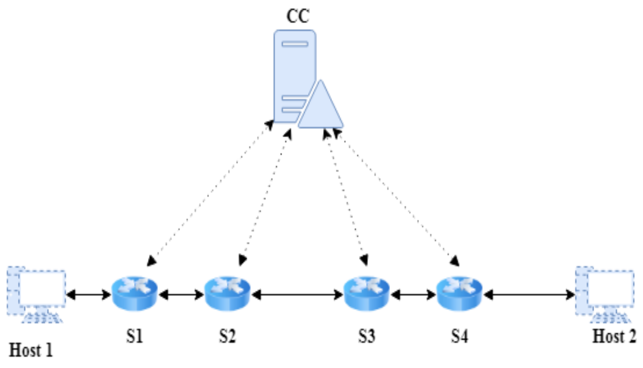


FIGURE 3. Centralized controller model.

has the following four key innovations over the traditional networks:

- *SDN* removes the control logic from network devices (e.g., routers and switches). So the network devices have become simple forwarding elements.
- *SDN* puts control logic in an external and central unit called *SDN* controller or Network Operating System *(NOS)*, which is a software that runs on a server. It obtains the global view of the network and make necessary routing decisions for the simple forwarding devices [38].
- *SDN* makes routing decisions *per* flow[1] while traditional IP routing is destination based. SDN then places identical policies on the forwarding devices in the determined path so that each packet of the flow receives consistent services [40]. This flow-based programming capability of SDN provides extraordinary flexibility in managing the network [7].
- *SDN* makes it easy to develop several new services by running new applications on top of *SDN* controller. Such applications simply get the abstract network view (*ANV*) from the *SDN* controller. Upon making the necessary policy decisions, they interact with the underlying data plane through *SDN* controller that maps higher-level policy decisions to lower-level forwarding decisions. This is considered as the most important innovation of *SDN*.

[1]A flow can be defined as a sequence of packets from a source to a destination [39].

B. CONTROLLERS IN SDN

In this section, we first explain three control models of *SDN*: Centralized, Hierarchical, and Flat.

CONTROL MODELS OF SDN*:*

- *Centralized*
  Control plane is initially created based on a centralized solution where one controller as seen in Figure 3 obtains the global view of the network and makes all the control decisions. While this design could be acceptable in small scale networks, it would have significant drawbacks as it creates a single point of failure and performance bottleneck for the network. To avoid single point of failure, we can use backup controllers. As a matter of fact, to be able to backup the state information at another controller, OpenFlow allows the connections from multiple controllers to a switch [7], [41]. However, we still need to deal with the performance bottleneck problem and other limitations of a single centralized controller. To overcome these limitations, researchers have proposed hierarchical and flat control models that we discuss next.
- *Hierarchical*
  In hierarchical approach, there are at least two levels of control as seen in Figure 4. At the bottom level, we have multiple local controllers while having a centralized controller at the top level. Local controllers respond faster and provide data-path services for the requests that need only local network state information. They redirect only the requests that require the global network view to the centralized controller [42]. This approach reduces overload on the centralized controller; but it still has the single failure of point problem [43].
- *Flat*
  In Flat approach, there are many physically distributed controllers that operate in a logically centralized manner as in Figure 5. These controllers use their local view to set up paths through their own domain. However,

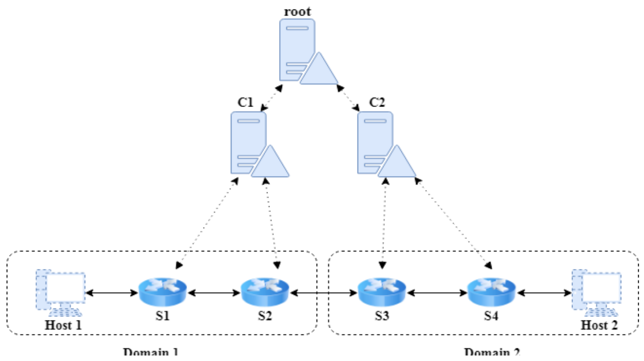


FIGURE 4. Hierarchical controller model.

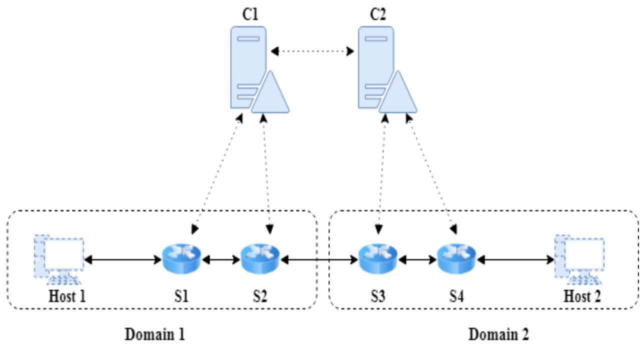


FIGURE 5. Flat controller model.

they also exchange their information with each other to deal with paths involving multiple domains. Accordingly, this approach has an advantage as it decreases the protocol overhead when compared to single centralized controller [44]–[46].

Controller Placement: Distributed mode of control avoids the single point of failure and provides performance improvements. However, it introduces several new challenges. For example, the number and placement of controllers are very important design issues and can significantly impact the performance. The number of *SDN* controller is critical in order to provide reliable network operation. Placement of these controllers is very important for network objectives such as inter-controller latency, reliability, fault tolerance, switch-controller latency, application requirements and path reliability [41], [43], [47]–[51].

The *SDN* placement problem was firstly presented in [47]. In [47], the authors investigated how many controllers needed and where to place them with respect to latency constraint and the network topology. They showed that the latency from every node to the controller can satisfy response time goals of existing applications in many medium-size networks with a single controller [52]. In [53], the authors studied the same problem with additional metrics such as load balancing among controllers and inter-controller latency. After discussing different resilience aspects, it was observed that there is not only one solution for different metrics, but a trade-off between them. In [54], authors provide a tool set which is very fast in facilitating the analysis and optimization of the controller placement in small and medium *SDN* networks under dynamic conditions. However, in the context of *SDN* based *WANs*, dynamically changing network state information dramatically affects performance of the network. Thus, the controller placement problem comes into prominence in order to control the dynamic changes in the *WANs*. So faster heuristic approaches are presented in [55]. Yet, even if heuristic approaches have faster computation time, they could sacrifice accuracy or vice-versa. Thus, the controller placement problem is still an open research challenge in *SDN*-based *WANs*.

C. ROUTING FRAMEWORK IN SDN

There are three common tasks for routing in SDN. The SDN controller needs to (*i*) discover network topology and obtain current statistics of links, (*ii*) compute ideal paths for the given flows, and (*iii*) use *OpenFlow* protocol to install forwarding rules on the SDN switches over the computed path.

After briefly discussing these tasks, we mainly concentrate on the tasks (*i*) and (*ii*) in the remainder of the paper. We examine the performance of various routing algorithms with two different NSI collection methods.

1) ACQUIRING THE GLOBAL VIEW OF THE NETWORK

In SDN, unlike classic routing protocols (e.g., OSPF), forwarding elements do not have any information about the network because the management plane is removed from them to a remote controller. Therefore, to perform routing decisions and compute new paths, the controller needs to discover network topology and *accurately* acquire the link state information from the underlying forwarding elements (e.g., routers and switches).

After the SDN controllers discover the network topology, they query the underlying switches and obtain the link-state information about each link. Using topology discovery protocols is sufficient for obtaining the *static* link state metrics because they do not change over time [8], [46]. On the other hand, since the *dynamic* link-state metrics (e.g., available bandwidth, utilization, delay, jitter) frequently change, the controllers need to periodically query the state information of each link to retain accurate NSI. In order to meet the flow demands and avoid congestion, we need to better utilize underlying network resources. Therefore, we have to obtain high accuracy in the link-state information. To do that, the controllers need to query the switches at very short intervals which causes significant overhead.

So *achieving the accurate acquisition of the dynamic link-state information while minimizing the overhead on the controller and in the network* becomes one of the challenging issues in SDN. To address this issue, further research is needed. In this study, however, we focus on the existing algorithms and evaluate their performance under idealistic case where we assume that the accurate NSI is available and practical case where the controller collects NSI periodically. We will try to mimic the availability of accurate state information by maintaining a shadow topology where the controller updates NSI after each path installation. Unfortunately, this method cannot be implemented in case of the involvement of multiple controllers, elastic background traffic, and excessive delays, etc. In such cases, the controller must periodically query each node and obtain the link-state information. Therefore, we will also evaluate the performance of the existing algorithms with periodic link-state collection method. We collect NSI with three different intervals and show their impacts on accuracy and the routing algorithms.

We discuss collection mechanisms and the details of our link-state representation in Section IV.

2) COMPUTING FEASIBLE AND/OR OPTIMAL PATHS

Using the NSI obtained in the first step, the SDN controller executes a routing algorithm to compute an ideal path for a given flow. We will discuss the routing algorithms in Section III.

All the existing routing algorithms need the underlying NSI to be represented as a directed graph $G = (V, E)$, where V is the set of nodes/switches and E is the set of links. Let $n = |V|$ be the number of nodes and $m = |E|$ be the number of edges in the network. Each link $(u, v) \in E$ is has a link cost denoted by $C(u, v)$. This cost is computed with respect to the acquired static or dynamic link-state information. Actually, as we further present in Section III, the difference between most of the existing algorithms is in how they determine $C(u, v)$ metric. Most of the existing routing algorithms generally use Dijkstra's shortest path algorithm or its modified versions to compute a path based on the determined link cost. In some algorithms, the links that do not meet the requested demands are eliminated before computing the paths.

3) INSTALLING THE FORWARDING RULES

After computing a path, the next step is to install the determined rules on each switch in the computed path. The SDN controller uses *OpenFlow* to install these rules on the forwarding table of each switch. The path computation and the rule installment process can be done in proactive, reactive, and hybrid modes. These three modes of operations can be described as follows:

Reactive Mode:

When a switch receives a request that does not match with the existing rules in its forwarding table, it considers this as a new flow and creates an Open Flow Protocol (*OFP*) packet-in message for the first packet of the new flow. It then sends this message to the controller for a decision. Accordingly, the controller creates a rule and sends it to the switch via OpenFlow. The switch uses this rule to forward future packets. Even though this approach uses existing flow table memory more efficiently, unfortunately, it causes performance delay because of continuous communication between forwarding elements and the controller for the first packet of each new flow. This delay can be ignored in small scale networks; but, it is a significant burden for geographically remote controllers, short-lived flows, general purpose CPU and/or vSwitches [41], [56]–[60].

Proactive Mode:

The SDN controller determines forwarding rules for possible traffic in advance and installs the necessary rules to the switches ahead of time. So when a new flow arrives, the switches can forward its packets without consulting with the controller. Accordingly, the switches forward all the packets at line rate [41], [56]–[58], [61], [62]. Moreover, since all flows find a match in the switches and avoid consultation with the controller, this mode significantly reduces the burden on the controller. The disadvantage of this approach is that the flow tables should be coarse-grained because of scalability issues. To meet fine-grained control needs, we can use the hybrid approach discussed next.

Hybrid Mode:

The controller and switches can use the combination of reactive and proactive modes. Accordingly, we can get the flexibility of reactive mode in providing fine-grained control while benefiting from proactive mode by eliminating delay and avoiding significant burden on the controller. Based on the network performance, the proactive mode can periodically modify routes and flow tables to improve performance [56]–[58].

Since we mainly evaluate the routing algorithms based on their performance in finding the paths for given flows, reactive mode is more suitable for our experiments. When a new flow is seen, the source switch will inform the controller. Accordingly, the controller will compute an appropriate path and install it through the network.

III. EXISTING ROUTING ALGORITHMS

We partition the existing routing algorithms into three categories: *RA-SLC* (Routing Algorithms with *Static* Link Cost), *RA-DLC* (Routing Algorithms with *Dynamic* Link Cost), and *RA-DLCMI* (Routing Algorithms with Dynamic Link Cost and Minimum Interference).

With some exceptions, almost all these algorithms have the same key steps. Therefore, before mentioning the details of each algorithm, we like to present the generic framework in Algorithm 1.

RA-SLC do not need the Steps 1 and 2 because they do not include current link-state information to determine cost metric for path computation. Therefore, *RA-SLC* use static link cost metric that results in computing same path for all the flows between a given pair of nodes (s, d). Conversely, *RA-DLC* and *RA-DLCMI* consider current link state information to determine cost metric for path computation. Accordingly, first two steps (especially, Step 1) become very important components for the algorithms. Actually,

Algorithm 1 Generic Framework for All Routing Algorithms

```
Input: Graph $G(V, E)$, vector $BW$ which is the initial bandwidth capacity of the links, vector $RBW$ which is the residual bandwidth of the links, vector $I$ which is the number of flows carried on the links, and a flow request between the pair of nodes $(s, d)$ with the required demand of $r_{(s,d)}$.
Output: $p$, a path between the pair of nodes $(s, d)$ that satisfies the requested demand of $r_{(s,d)}$. That is $\min\{RBW_{(u,v)} | (u, v) \in p\} \geq r_{(s,d)}$.
1: Based on the proposed heuristic, compute link cost metric $c_{(u,v)}$ for all edges $(u, v) \in E$.
2: Eliminate all links that have residual bandwidth less than requested demand $r_{(s,d)}$.
3: Use Dijkstra's algorithm to compute the shortest path $p$ between the pair of nodes $(s, d)$.
4: SDN Controller installs the forwarding rules on the SDN switches in $p$.
5: SDN switches route the packets of the requested demand $r_{(s,d)}$ through path $p$.
6: SDN Controller periodically collects the updated link-state information.
```

the key difference among all these algorithms is in how they determine the dynamic link cost metric in Step 1. Step 2 is the same for all algorithms; they simply eliminate the links which do not have enough residual bandwidth capacity to carry the requested demand. We should also note that *Dynamic Widest Shortest Path (DWSP)* under RA-DLC computes a path with the largest residual bandwidth in Step 3 by using a modified version of Dijkstra's algorithm that directly uses the residual bandwidth of each link. So it does not need to use Step 1.

We consider the first two steps as the pre-computation phase for *RA-DLC* and *RA-DLCMI*. For each new flow, this pre-computation phase is performed because of changes in the link-state information after installing the forwarding rules for each previous flow. However, since acquiring the up to date link-state information of all links is very costly, the SDN controller periodically collects that information in practice. However, as we examine later in detail, the periodic collection causes some inaccuracy in the link-state information and negatively affects the performance of the routing algorithms.

The last four steps are the same for all *RA-DLC*, *RA-DLCMI*, and *RA-SLC*. In Step 3, all algorithms use some form of *Dijkstra's algorithm* to compute the best path p (e.g., least cost path, widest path) based on the dynamic or static link cost obtained in Step 1. In step 4, the SDN controller installs the necessary rules on the SDN switches in the computed path p. Then, in Step 5, the packets of the given flow are routed through p. Lastly, in Step 6, in order to make routing decisions for the new flows, the SDN controller obtains the dynamic link-state information (e.g., residual bandwidth) by periodically inquiring all switches in the network.

In the next three subsections, We discuss the key assumptions and the ideas behind the aforementioned existing routing algorithms that we categorized into three classes, as shown in Table 1. Besides, we investigate the advantages and disadvantages of each group of the routing algorithms and summarize their key characteristics.

A. *ROUTING ALGORITHMS WITH* STATIC *LINK COST* (RA-SLC)

For *RA-SLC*, the SDN controller determines the static link cost (e.g., hop count, distance, 1/bandwidth) for each link *once* while discovering the network topology. The SDN controller does not need to obtain dynamic link-state information (e.g., link utilization) because *RA-SLC* do not consider changes in link-state information.

Clearly, since the SDN controller does not need to periodically collect current link-state information or re-compute the link cost with respect to the current link-state information, the algorithms that use static link cost are more useful in terms of reducing the message and the pre-computation overhead. Although this approach might perform better when the network utilization is low, it will cause serious congestion and unbalance when the network utilization increases because it always finds the same paths for the flows with the same source and destination pairs.

To overcome these problems, researches have proposed *RA-DLC* and *RA-DLCMI* that we discuss in the next section. To better evaluate the performance improvements of *RA-DLC* and *RA-DLCMI*, we have included the following *RA-SLC* in our evaluations.

Minimum Hop Algorithm (MHA) computes the shortest path with the minimum number of links between source and destination nodes [30]. To do that, the algorithm sets the cost metric of each link to 1 and runs BSF (Breadth First Search) or *Dijkstra's Algorithm*.

Shortest Path Algorithm (SP) sets the cost metric of each link to inversely proportional to the bandwidth capacity of that link and runs *Dijkstra's Algorithm* to find the shortest path based on the cost metric [28]. This approach is also used in *Open Shortest Path First (OSPF)* [64].

Widest-Shortest Path Algorithm (WSP) uses a modified version of Dijkstra's algorithm that aims to compute a path with the largest bandwidth capacity and minimize hop-count [29], [65]. There is another version called Shortest-widest path algorithm (SWP) that first eliminates all the links that do not have enough capacity to carry the given demand request and then uses the Dijkstra's shortest path algorithm to select the links with the largest bandwidth capacity [66].

B. *ROUTING ALGORITHMS WITH* DYNAMIC *LINK COST* (RA-DLC)

Even there may be alternative links to use, *RA-SLC* finds the same paths for all the given flows between the same source and destination nodes that results in congestion on some links. To overcome this issue, the controller periodically collects dynamic NSI (e.g., available bandwidth, link utilization)

TABLE 1. Classification of existing routing algorithms.

Type of Routing Algorithm	Algorithm Name	Cost metric calculation	Computation Complexity	
			Pre-computation per flow	Path Selection Algorithm per flow
RA-SLC	*MHA*	$C_{(u,v)} = 1$	-	Dijkstra's Algorithm
	SP	$C_{(u,v)} = \frac{1}{BW_{(u,v)}}$		
	WSP	$C_{(u,v)} = BW_{(u,v)}$		*Modified Dijkstra's Algorithm*
RA-DLC	*DSP*	$C_{(u,v)} = \frac{1}{RBW_{(u,v)}}$	$\mathcal{O}(m)$ for computing cost metric for each link	*Dijkstra's Algorithm*
	DWSP	$C_{(u,v)} = RBW_{(u,v)}$		*Modified Dijkstra's Algorithm*
RA-DLCMI	*LIOA*	$C_{(u,v)} = \frac{I^{\alpha}_{(u,v)}}{RBW^{\alpha}_{(u,v)}}$		Dijkstra's Algorithm
	ILIOA	$C_{(u,v)} = (1 - U_{(u,v)}) * \frac{I^{\beta}}{BW^{\beta}_{(u,v)}} + U_{(u,v)} * \frac{I^{\alpha}}{RBW^{\alpha}_{(u,v)}}$		
	MIRA	Initialize $C_{(u,v)}$ to 0. Using max-flow min-cut algorithm, find min-cut sets (critical links) between the pairs. For each appearance of a critical link (u, v) in the min-cut sets, increase $C_{(u,v)}$ by 1.	$\mathcal{O}(m)$ for computing cost metric for each link + $\mathcal{O}(max - flow\ min - cut)$ for finding critical links (e.g., $\mathcal{O}(n^2\sqrt{m})$ with Goldberg-Tarjan highest label pre-flow push algorithm [63])	
In this table, $C_{(u,v)}$ represents cost metric, $BW_{(u,v)}$ is the initial bandwidth capacity, $RBW_{(u,v)}$ is residual bandwidth capacity, $I_{(u,v)}$ is number of flow carried on the link (u, v). $U_{(u,v)}$ is equal to $1 - RBW_{(u,v)}/BW_{(u,v)}$. m is the number of the links and n is the number of nodes in the topology. We choose $\alpha = 0.5$ and $\beta = 0.3$ in our tests because they are shown to give the best results [32], [33]				

and uses this information to compute paths. Accordingly, researchers have proposed various routing algorithms with *dynamic* link cost to better utilize the network resources and increase the throughput by preventing congestion. The main goal of these algorithms is that minimizing the rejection of future demands while finding paths for the requested demands.

In on-line setting that the flows arrive one at a time, it is not possible to achieve these goals completely because future demands are not known in advance. Furthermore, the problem is *NP-Complete* even the flows are known in advance (off-line setting) [36], [67]–[70]. Therefore, researchers have proposed heuristic algorithms. We present the basic forms of these proposed heuristic algorithms as *RA-DLC* in this subsection and their advanced forms as *RA-DLCMI* in the next subsection.

Actually, *RA-DLC* take into account *residual* bandwidth of each link instead of *static* link cost and use the aforementioned *SP* and *WSP* to compute paths. Correspondingly, we name these algorithms as **Dynamic Shortest Path *(DSP)*** and **Dynamic Widest-Shortest Path *(DWSP)***, respectively [29], [31].

RA-DLC are able to find different paths for the flows between the same source and destination nodes. This is the main advantage of RA-DLC because they better utilize network resources and maximize its revenues. However, in order to accurately acquire the dynamic link-state information, these algorithms cause protocol overhead. Furthermore, these algorithms need to re-compute the link cost with respect to current link-state information which causes extra pre-computation cost before starting to compute a path for the new flow.

C. *ROUTING ALGORITHMS WITH* DYNAMIC *LINK COST AND MINIMUM INTERFERENCE* (RA-DLCMI)

In order to increase the chance of accepting more future demands and optimizing performance (e.g., less delay and loss), the dynamic link metrics (e.g., residual bandwidth) used in *RA-DLC* may not be enough. Accordingly, *RA-DLCMI* try to minimize the *interferences* among the flows by evenly distributing the traffic load through the network.

To accomplish these goals, *RA-DLCMI* determines the cost of each link by using the other metrics (e.g., critical links between source-destination pairs and/or *number of flows* carried on the links) along with the dynamic cost metrics (e.g., residual bandwidth).

In the literature, researchers have proposed several heuristic algorithms by using different combination of various metrics to determine the dynamic cost of each link. They mainly differ in how they determine the dynamic cost of each link. In this subsection, we first present the details of the algorithms that we implemented. Then, we briefly present the other proposed heuristic algorithms.

Minimum Interference Routing Algorithm *(MIRA)* considers that all the demands flow through a pre-defined set of (ingress-egress) pairs. *MIRA* tries to determine crucial links for each pair. So, it uses the links that are less crucial to the other pairs when computing the shortest path for a given flow between a pair [30].

Accordingly, it initializes the cost metric of each link to 0 before computing the shortest path for a given pair. *MIRA* finds the min-cut set between each pair except for the pair for which it tries to find a path. To do that, it uses the residual bandwidth information in the *max-flow min-cut* algorithm. It adds 1 to the cost of each link if it is in min-cut sets. Then, it runs Dijkstra's algorithm based on the cost metric to compute the shortest path. *MIRA* maximizes the minimum available capacity between all other pairs. However, because of its significant computational complexity of the pre-processing step for each flow, it is not an effective algorithm to use in practice, especially if the network is large and the number of pairs is high.

Least Interference Optimization Algorithm *(LIOA)* uses the inversely proportional ratio of the number of flows on each link to the residual capacity on that link to determine the link cost metric [32]. Accordingly, *(LIOA)* computes the link cost using $C_{(u,v)} = \frac{I^{\alpha}_{(u,v)}}{RBW^{\alpha}_{(u,v)}}$, and then runs Dijkstra's algorithm to find the shortest path. As it should be, *(LIOA)* chooses the links with less number of flows and high residual bandwidth. Accordingly, it balances the number and quantity of flows in the network to reduce the interference between source-destination (ingress-egress) pairs.

Improved Least Interference Optimization Algorithm *(ILIOA)* is modified version of *LIOA* and takes more metrics into account in computing the link cost [33].

(ILIOA) uses the number of flows on the links, initial bandwidth capacity, residual bandwidth capacity, and utilization of the links.

Specifically, *(ILIOA)* computes $C_{(u,v)} = (1 - U_{(u,v)}) * \frac{I^{\beta}}{BW^{\beta}_{(u,v)}} + U_{(u,v)} * \frac{I^{\alpha}}{RBW^{\alpha}_{(u,v)}}$. If link utilization is low, $(1-U_{(u,v)}) * \frac{I^{\beta}}{BW^{\beta}_{(u,v)}}$ dominates the cost metric to choose the links with large capacities. Otherwise, $U_{(u,v)} * \frac{I^{\alpha}}{RBW^{\alpha}_{(u,v)}}$ aims to choose the links with large remaining bandwidth.

In addition to the aforementioned algorithms, there are some other *RA-DLC* and *RA-DLCMI*. For example, Dynamic Link Weight (*DLW*) avoids congested links by trying to choose lower loaded paths [71]. As in *MIRA*, Wisitak's Routing Algorithm (*WSS*) determines a cost parameter to detect possible critical links [72]. Unlike, it does not consider other dynamic link costs such as residual bandwidth capacity. Fabio's Weight Function (*WF*) uses dynamic link costs; but, it does not take the use of pair interferences [73]. BU-MIRA takes into account the available bandwidth of each link and the number of flows carried on it when computing the cost metric for each link [74]. The algorithm by Wang-Su-Chen (*WSC*) is the extended version of *MIRA* and uses a different method to determine the cost metric [75]. Bandwidth Constrained Routing Algorithm (*BCRA*) takes into account the residual capacity, initial bandwidth capacity of the links, and the number of the flows carried on the links to detect interference, as in *LIOA* [76]. Maximize Residual Bandwidth and Link Capacity - Minimize Total Flows Routing Algorithm (*MaxRC-MinF*) determines a cost metric as in *LIOA* and *BCRA* [77]. We will not include these algorithms in our evaluations because they have been intensively evaluated and shown that the three algorithms (MIRA, LIOA, ILIOA) that we implement have outperformed them. In addition, we did not include Light Minimum Interference Routing (*LMIR*) [69] algorithm in our evaluations because of some ambiguities in some steps that did not lead to finding the lowest capacity paths.

IV. ACQUIRING NETWORK STATE INFORMATION

All the algorithms we presented have been proposed with the assumption that the controller has the accurate network state information (NSI) before computing a path. Unfortunately, the controller needs to continuously query the distributed switches to obtain the accurate NSI. However, this will cause significant protocol overhead and obtained NSI may not be accurate because of delays and measurement errors. In order to reduce protocol overhead and obtain accurate NSI, the controller uses a periodic monitoring method that collects the link-state information from each switch at a predetermined rate (e.g., every $\mathscr{T}$ seconds) [10]. Selecting the value of $\mathscr{T}$ is very crucial because if it is reduced, the controller gets considerably more accurate NSI at the expense of increasing protocol overhead.

In the remainder of this section, we will firstly discuss how the availability of accurate NSI can be mimicked at the controller. Then, we present the main steps of periodically collecting NSI. In the next section, we use both methods (accurate NSI and periodically collected NSI) to better evaluate the existing algorithms under the ideal and practical settings, respectively.

1. **Accurate Network State Information (*NSI*):** The controller keeps the shadow of the topology by updating the residual link capacities after installing a flow to mimic the accurate NSI availability. The controller computes a path for a given flow demand. Then, it subtracts the requested demand from the residual capacity of each link on the computed path. Therefore, the controller will always have current residual graph to compute a path for a new request. Clearly, we can use this idealistic approach if there is only one physical controller and there is no background traffic as we implemented our simulation. Therefor, we can use this method to better compare the performance of the algorithms under the common assumption of having accurate NSI.
2. **Periodically collecting *NSI*:** In practice, to acquire NSI periodically, the controller sends the port statistics requests to the all *OpenFlow* switches in every $\mathscr{T}$ seconds. Upon receiving this request from the controller, each switch sends the total amount of traffic sent on its ports until the request comes. Let us keep the following quantities at the controller: $tx^{c}_{(u,v)}$ be the total amount of traffic sent on link (u, v) until the current time t_c, and $tx^{p}_{(u,v)}$ be the previously recorded total amount of traffic sent on link (u, v) until the previous query time t_p. Using these quantities, the controller can update the residual bandwidth of link (u, v) as follows:

$$\mathcal{D}_{(u,v)} = \frac{tx^{c}_{(u,v)} - tx^{p}_{(u,v)}}{t_c - t_p} \tag{1}$$

$$RBW_{(u,v)} = BW_{(u,v)} - \mathcal{D}_{(u,v)} \tag{2}$$

From the equation (1), we determine the average current traffic rate denoted by $\mathcal{D}_{(u,v)}$. From the equation (2), we determine the residual bandwidth $RBW_{(u,v)}$ by subtracting $\mathcal{D}_{(u,v)}$ from the $BW_{(u,v)}$, which is the initial capacity of the link. Finally, the controller uses the following equations to save the current quantities as the previous quantities for the calculations in the next period.

$$t_p = t_c \tag{3}$$

$$tx^{p}_{(u,v)} = tx^{c}_{(u,v)} \tag{4}$$

V. PERFORMANCE EVALUATION

In this Section, we compare the performance of several existing routing algorithms that we implemented using *RYU SDN* Controller [10]. *RYU* allows us to create new control applications by providing application program components and/or interfaces (*API*). We then run the controller on *Mininet* emulator, which enables us to create a network of virtual switches, hosts, and links [11], [78]. OpenFlow protocol (version 1.3) is used to communicate with the forwarding elements on *Mininet*. We use *Linux* software and *iperf* for performance measurements and the traffic generation at the hosts on *Mininet* [79].

Since *TCP* flows aim to maximize throughput by using all available bandwidth on the links so this causes misleading to acquire better evaluation for the performance of the routing algorithms which consider the given requested demand. Therefore, instead of *TCP*, we generated *UDP* flows with uniformly assigned demands. When a randomly generated flow request $(s, d, r_{(s,d)})$ is received in the controller, a path from source s to destination d with demand of $r_{(s,d)}$ is computed by using one of the routing algorithms.

If the controller finds a feasible path where each link on the path has enough available capacity to carry the demand, it installs the essential forwarding rules to the switches through the path. Otherwise, it rejects the flow. When a flow is established, it continuously loads the network with its demand until the end of the simulation.

We use the following performance measures to evaluate the performance of the algorithms by testing each algorithm with the same set of flow requests:

Number of Accepted Flows (NAF)
The number of flows for which the algorithm is able to find a feasible path.

Total Transferred Bandwidth (TTB)
The total amount of the data rate transferred for the accepted flows, which might be smaller than the total demand of the accepted flows as some of their packets could be lost.

Packet Loss (PL)
The percentage of the all the sent packets got lost when being transferred through the network.

Normalized Path Computation Time (NPCT)
The normalization of the computation time of each algorithm over the computation of the algorithm which has the minimum computation time.

In the remainder of this section, we present the network topologies used in our experiments. We used both NSI collection methods that we mentioned in Section IV. Accordingly, we first show the performance evaluation results under the idealistic method. Then, we used *Periodically Collecting NSI* method with the intervals of $\mathscr{T} = 3, 5, 10$ seconds.

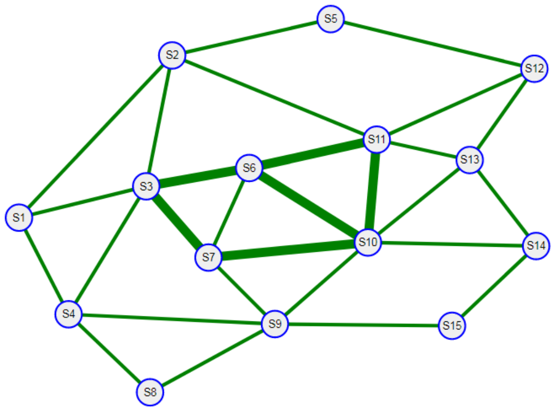


FIGURE 6. MIRANET topology.

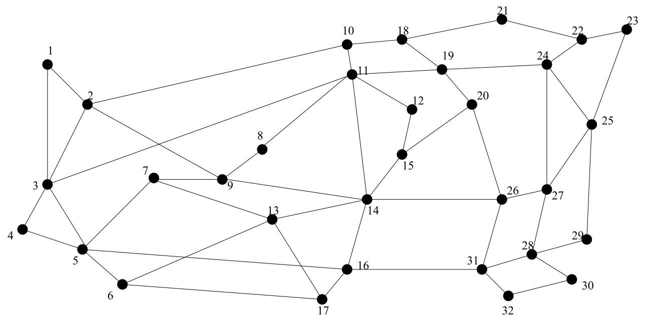


FIGURE 7. Extended ANSNET topology.

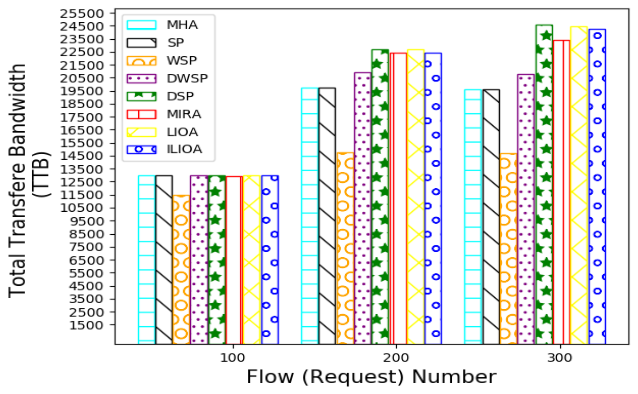
A. NETWORK TOPOLOGIES

We used *MIRANET* [30] and extended *ANSNET* [80] topologies to compare the routing algorithms.

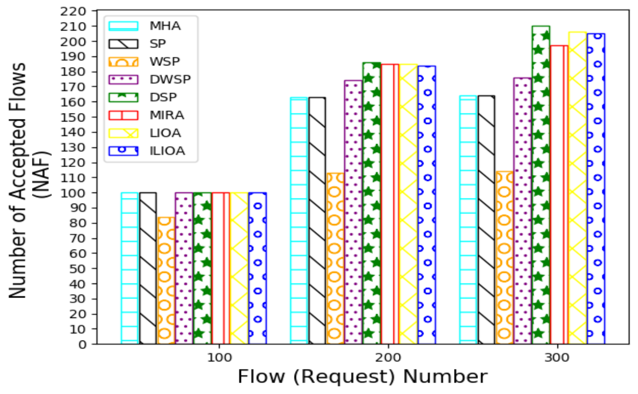
As seen in Figure 6, *MIRANET* topology contains 56 bidirectional links and 15 nodes. *MIRANET* is used to compare the algorithms in [30]–[33]. We assigned 5 Mbps to the thinner links and 25 Mbps to the thicker links 2. As originally declared in [30]–[33], we used the same four (s, d) pairs (namely, $(S1, S13)$, $(S5, S9)$, $(S4, S2)$ and $(S5, S15)$). We generated 100, 200, and 300 flows with requested demand of *uniform*(50, 200) kbps.

We also used the realistic network topology extended from *ANSNET* in [80] to evaluate the performance of the algorithms on a larger topology. *ANSNET* topology has 32 nodes and 108 bi-directional links. The bandwidth capacity of each link is randomly selected from *uniform*(2, 10) Mbps.[2] We randomly selected the source from the leftmost nodes (1, 2, 3, 4, 5) and the destination from the rightmost nodes (23, 25, 29, 30, 31). At this time, we match the pairs between any source and destination which results in 25 different possibilities. Since *MIRA* has to run *max-flow* algorithm for all pairs (excluding the pair which we try to compute a path for it) its computation time increases significantly under larger topologies, we did not include *MIRA* algorithm to evaluate under this setting. We randomly generated 50, 75, and 100 flows with requested demand of *uniform*(200, 500) kbps.

[2]In Mininet, the forwarded packets share CPU and memory resources of the computer, resulting in significant overload and limitations [81]. To overcome these limitations, particularly on large topologies, we have to use smaller link capacities and less number of flows to reach reasonable load in the network.



(a) Throughput (MBytes) under MIRANET with Accurate NSI



(b) Accepted Flows under MIRANET with Accurate NSI

FIGURE 8. Throughput and number of accepted flows under *MIRANET* topology with accurate NSI.

B. PERFORMANCE RESULTS UNDER ACCURATE NSI

Since using *dynamic* link costs allow the *RA-DLC* and *RA-DLCMI* to compute alternative paths for the flow requests, they outperform the *RA-SLC* (e.g., MHA, SP, WSP) in terms of the total transferred bandwidth (TTB) and the number of accepted requests (NAF).

We mainly focus on the performance of *RA-DLC* and *RA-DLCMI*. As we see in Figure 8, *DWSP* gives worse performance than *DSP*, *LIOA* and *ILIOA* while they perform similarly. In contrast to the low performance under the MIRANET topology, *DWSP*'s performance under the ANSNET topology is similar to others, as we present later.

According to our observations, after larger bandwidth capacity links are occupied, *DWSP* cannot compute paths for the flow requests that have larger demands due to lack of alternatives in MIRANET topology.

Since *MIRA* runs the *max-flow min-cut* algorithm for all pairs, it is the worst algorithm among the three best algorithms in terms of normalized computation time. We first determine the computation times of the algorithms and then normalized them with the algorithm that has minimum computation time.

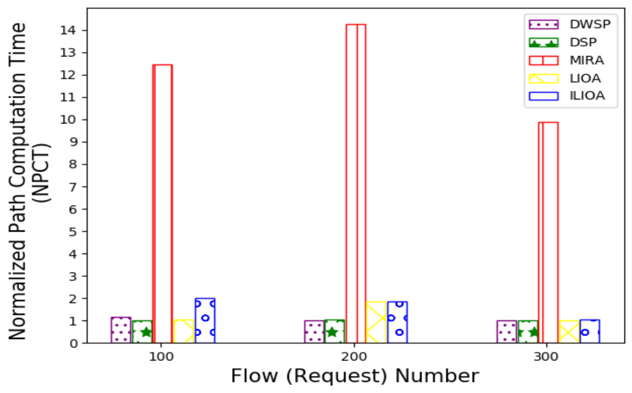
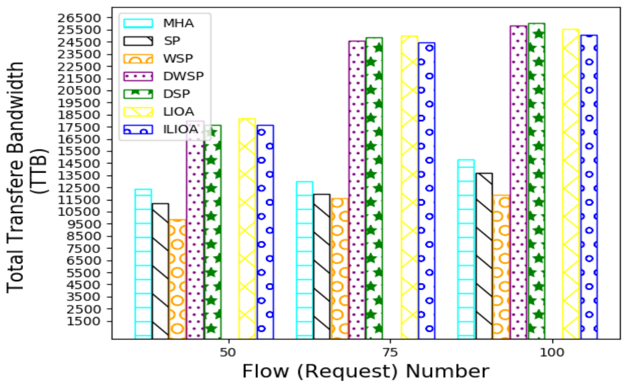


FIGURE 9. **Normalized average computation time under *MIRANET* topology with accurate NSI.**



(a) Throughput (MBytes) under ANSNET with Accurate NSI

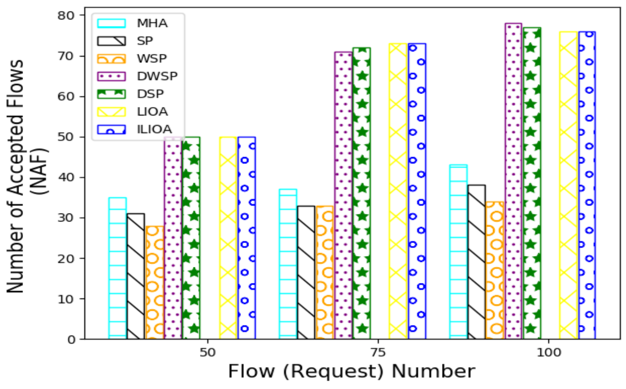


FIGURE 10. **Throughput and the number of accepted flows results under *ANSNET* topology with accurate NSI.**

As seen in Figure 9, Normalized Path Computation Time (NPCT) of *MIRA* is $10-15$ times worse than the others. Since the *RA-SLC* only use Dijkstra's algorithm without any pre-computation, they give similar performance to the fastest *dynamic* algorithm. Therefore, we did not include them in the NPCT evaluation.

Because of the excessive computation time of *MIRA*, we did not include it when we compare the other algorithms under the larger topology named ANSNET which has 25 source-destination pairs.

As we see in Figure 10, we again see that *RA-SLC* fall behind of *RA-DLC* and *RA-DLCMI* in the number of accepted flows and total transferred bandwidth. At this time, performance of *DWSP* is similar to other *RA-DLC* and *RA-DLCMI*. We think the reason is that ANSNET topology has more alternatives that the algorithms are able to explore.

During our experiments under accurate NSI, we observed that there is $1-2\%$ and $3-7\%$ packet loss under *MIRANET* and *ANSNET*, respectively. In theory, this should not happen because the algorithms reject a flow that has higher requested demand than the available link capacities. The delay in computing paths and installing the rules on the switches through the found paths cause some of the packet losses. Furthermore, OpenFlow Discovery Protocol (*OFDP*) causes the protocol overheads which the shadow topology and NSI we use at the controller do not take into account. In practice, in order to discover the topology, the controller interacts with the switches by sending encapsulated Link Layer Discover Protocol (LLDP) packets. Switches transmit packets to one of their adjacent switches by using a port. This switch responds the controller by encapsulating the received LLDP packet. This communication process grants the controller to learn there is a unidirectional link between the two switches. All links perform the same process in the network which may increase the usage of the controller and load on the links. Therefore, the cost of topology discovery increases linearly if the size of topology increases [82]. Since we ignored this process when maintaining the shadow topology to mimic accurate NSI, we have been exposed to the packet losses we mentioned above.[3]

Since the periodic collection method obtains the current total traffic including discovery protocol traffic on the ports, it is expected to avoid this kind of loss. However, as we discuss in the next subsection, we still observe packet loss under periodically collected NSI because the routing algorithms use inaccurate NSI and accept more flow than the network is able to carry.

C. PERFORMANCE RESULTS UNDER PERIODICALLY COLLECTING NSI

Now, we use *periodically collecting NSI* method to examine the routing algorithms under the same two topologies. We first present some general observations. Under periodic collection of NSI method, the controller cannot obtain accurate NSI because it needs to wait until the next period. Thus, since the information from the previous period is used, the requests admitted by the controller may exceed the current available bandwidth capacities of the links. This may cause that all flows may lose packets.

[3]In our simulations, we observed that we can minimize the packet losses if we assume $10-15\%$ of the link bandwidths are reserved for LLDP and compute packets based on this assumption. However, further research needed to determine optimal method

[4]Second, possibly some flows may stop sending traffic, resulting in decreasing load on the link. However, the controller will not obtain the updated information until the next query time. Therefore, assuming the link is still overloaded, it may reject some incoming flows, causing underutilization of the network.

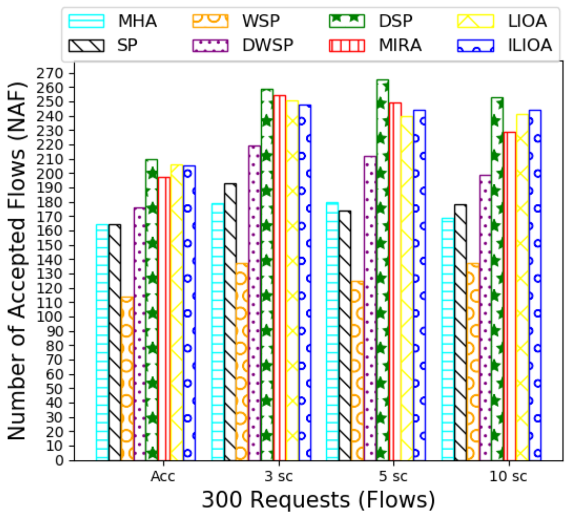
In this section, we used the same set of flows as we generated in section V-B and the same two topologies. As we discuss later in detail, when NSI is collected in longer periods, the number of accepted flows (NAF) increases, resulting in more packet loss (PL) that negatively affects total transferred bandwidth (TTB) as seen in Figure 13. We considered various intervals (short, medium, and large) for NIS collection and observed similar trends. Accordingly, to minimize simulation times, we used $\mathscr{T} = 3, 5$, and 10 seconds as the representative of short, medium, and large intervals for NSI collection. Due to similar trends we observed with the different numbers of flows, we only report the results with 300 flows under *MIRANET* topology and 100 flows under *ANSNET* topology.

Firstly, we present the simulation results under *MIRANET* topology. We observe that all the algorithms accept more flows under periodically collected NSI than that under accurate NSI, as seen in Figure 11(a). The reason for this is that the controller does not obtain current state information and wrongly overload some links by accepting more flows. As we see in Figure 11(b), this results in higher total throughput. However, as we see in Figure 11(c), accepting more flows than the network is able to carry causes each flow to lose huge number of packets. Obviously, this is not acceptable for the sake of providing Quality of Service (QoS).

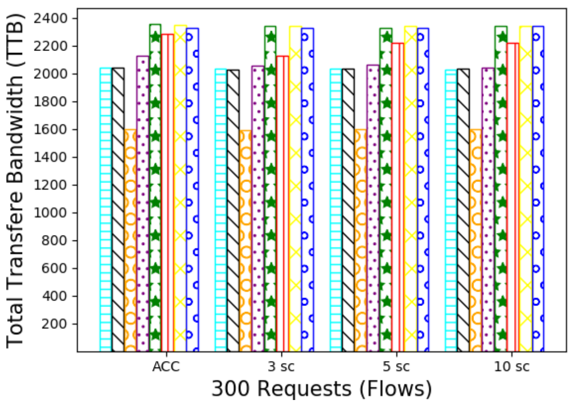
Naturally, we expect to have more accurate NSI, when we decrease the collection interval. In contrast, we realized that the algorithms accept more flows that result in more packet loss when NSI is collected in every 3 seconds than 5 or 10 seconds. We believe that there can be two reasons for this: (i) using shorter periods to collect NSI causes significant overload in the network which negatively affect the robustness of the controller, (ii) collecting NSI with shorter periods does not obtain enough statistical information to make accurate estimation. Although more accurate estimation is possible under 5 and 10 seconds collection, packet loss is still high since the controller use older and inaccurate statistical information.

We now present the simulation results under *ANSNET* topology. As observed above, all the algorithms again accept

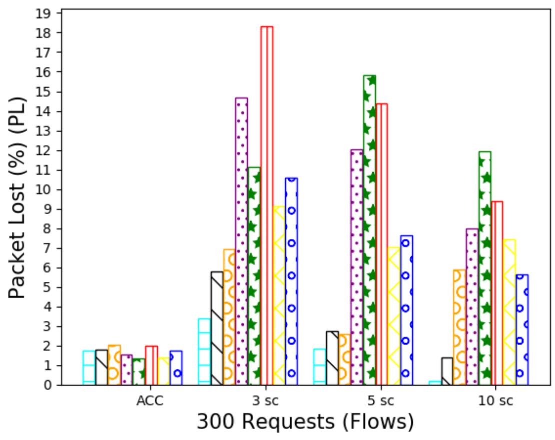
[4]We explain this situation with the following example. Let assume that there is a link that has 15 *Mbps* capacity and we generate two requests, each with 10 *Mbps* demands. In the case of accurate NSI, the algorithms accept first flow and transfer the flow on the link. So, the available bandwidth capacity of the link is updated as 5. Algorithms reject the second flow because the demand (10 *Mbps*) is higher than the remaining bandwidth capacity (5 *Mbps*) of the link. However, under periodically collecting NSI, since the controller did not obtain the updated information of the link yet, algorithms accept the second flow too. Certainly, overload on the link is inevitable. Thus, although both flows may give better or similar performance than a single flow under accurate NSI in terms of the throughput, both flows lose packets under periodically collecting NSI method.



(a) Accepted flows under MIRANET.



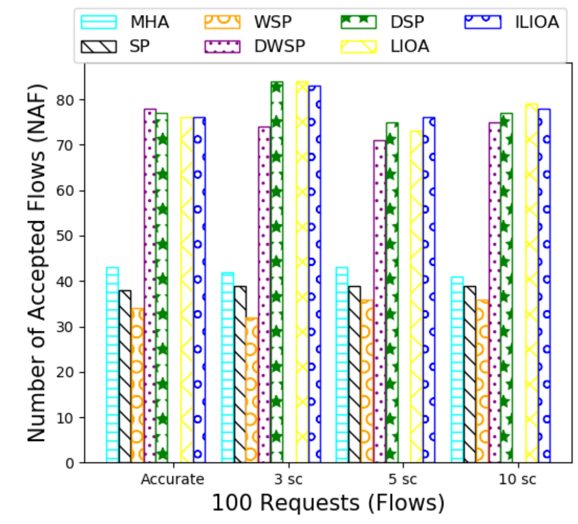
(b) Throughput (MBytes) under MIRANET.



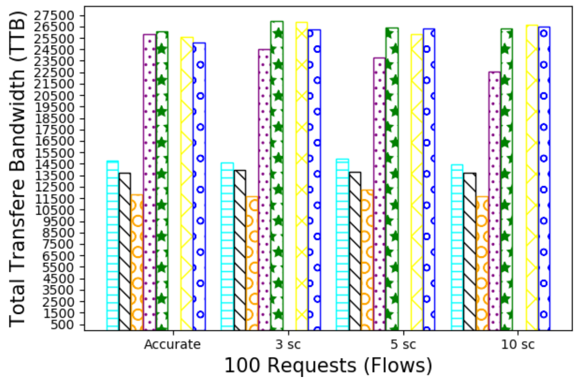
(c) Packet Lost (%) under MIRANET.

FIGURE 11. Accepted flows, throughput, and packet lost (%) under *MIRANET* topology with accurate and periodically collected NSI in every 3, 5 and 10 seconds.

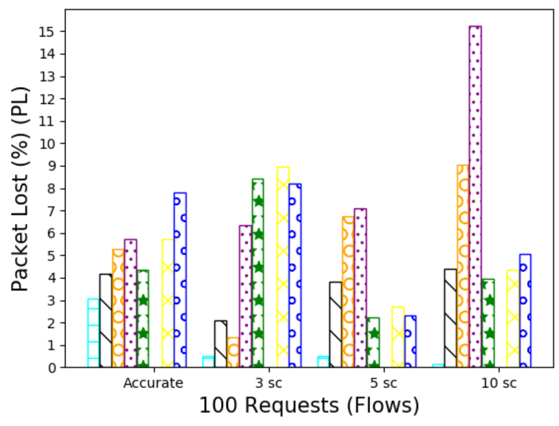
more flow requests when we periodically collect NSI in every 3 seconds, as seen in Figure 12(a). Accepting more flows provides higher total throughput at the cost of more packet loss, as we see in Figures 12(b) and (c), respectively.



(a) Accepted Flows under ANSNET.

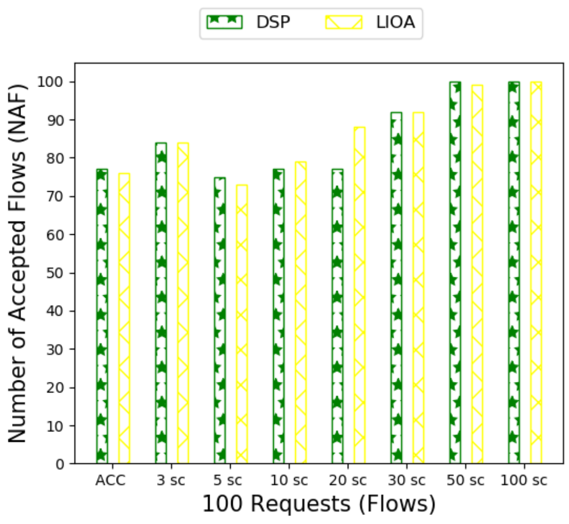


(b) Throughput (MBytes) under ANSNET.

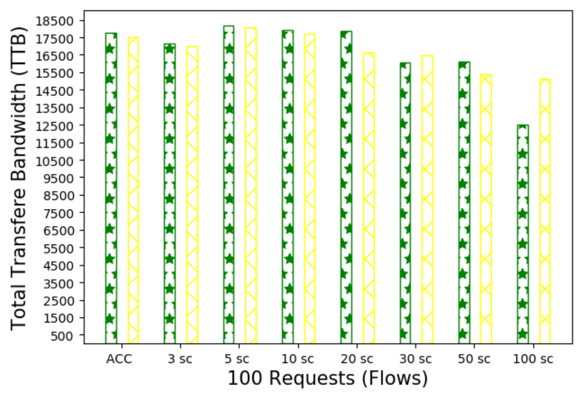


(c) Packet Lost (%) under ANSNET.

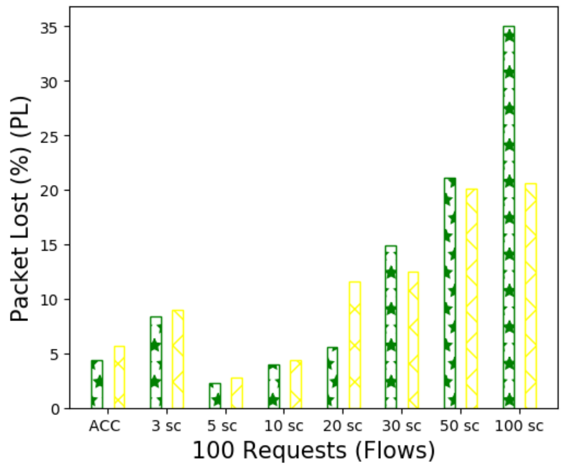
FIGURE 12. **Accepted flows, throughput, and packet lost (%) under *ANSNET* topology with accurate and periodically Collected NSI in every 3, 5 and 10 seconds.**



(a) Accepted Flows under ANSNET.



(b) Throughput (MBytes) under ANSNET.



(c) Packet Lost (%) under ANSNET.

FIGURE 13. **Accepted flows, throughput, and packet lost (%) under *ANSNET* topology with accurate and periodically collected NSI in every 3, 5, 10, 20, 30, 50 and 100 seconds.**

On the other hand, under the periodic collection in every 5 seconds, the algorithms accept fewer request, causing under-utilization of the network resources. When we increase the collection interval to 10 seconds, the packet loss increases more significantly. The reason should be that collecting NSI in longer interval time does not provide accurate state information, resulting in computing the same paths for the different requests.

In order to further evaluate how the length of collection period impacts performance, we now present the simulation results with periodically collected NSI in every 3; 5; 10; 20; 30; 50; and 100 seconds under ANSNET topology. As the representatives of RA-DLC and RA-DLCMI algorithms, we consider DSP and LIOA, respectively. As we see in Figure 13(a), both algorithms accept more flow requests when NSI is periodically collected in longer periods. In fact, in every 50 and 100 seconds, all flows are accepted by both algorithms. However, total throughput is decreasing and packet loss is increasing as seen in Figure 13(b) and (c). As we mentioned above, in the case of longer period of collection, algorithms wrongly use the same available bandwidth capacities of the links and calculate the same paths for the same source-destination flows regardless of their demands. Therefore, it is very important to determine length of period to collect NSI more accurately. This period could be vary based on the size and density of the topology.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we grouped the existing algorithms into three categories: (*a*) Routing Algorithms with Static Link Cost (*RA-SLC*) compute paths by using hop count, distance and/or link capacity which are static link costs that do not change during network traffic [28]–[30]; (*b*) Routing Algorithms with Dynamic Link Cost (*RA-DLC*) computes path by using dynamic link cost such as remaining bandwidth capacity of the links; (*c*) Routing Algorithms with Dynamic Link Cost and Minimum Interference (*RA-DLCMI*) are the extended version of the *RA-DLC* that compute paths by using other dynamic link costs such as link utilization, number of flows carried on the links together with aforementioned dynamic and static link costs [30]–[33].

In the context of SDN-based networks, we evaluated the performance of the existing routing algorithms. We used RYU SDN-controller to implement the algorithms. Then, we tested the implemented algorithms on Mininet emulator by using two different network topologies. The reason to choose *MIRANET* topology is that it is the main topology used for the algorithms in the literature. We also wanted to test the algorithm on a larger topology so we used extended *ANSNET* topology.

Obtaining network state information (NSI) was one of the key challenges. All routing algorithms have been proposed assuming that the controller has accurate NSI. However, it is a challenge to achieve high accuracy in practice because the controller periodically collects NSI. Therefore, we try to evaluate the existing routing algorithms under both the idealistic method where we maintain the shadow of the network topology to mimic accurate NSI in the controller and the practical method where the controller collects NSI periodically sacrificing some accuracy.

As we expected, we observe that *RA-DLC* and *RA-DLCMI* outperform the *RA-SLC* in terms of the throughput and accepted requests under both topologies. There is no significant difference among *RA-DLC* and *RA-DLCMI* algorithms but *MIRA* had very excessive computation time compared the others. Under accurate *NSI*, we were not expecting any packet loss. However, we have detected $\%1 - 2$ packet loss in *MIRANET* topology and $\%3 - 7$ packet loss in *ANSNET* topology. We believe that it happens because of two reasons: (*i*) the background traffic generated by OpenFlow Discovery Protocol (OFDP) during network topology discovery and (*ii*) the delays until the paths are installed.

There was no consistence in results because of inaccuracies when we tested the algorithms under periodically collected *NSI* in every 3, 5 and 10 seconds. When the controller cannot have up to date link-state information, the algorithms use the old and inaccurate statistical information. This causes the algorithms to wrongly accept more flows than the network capacity, resulting in significant packet loss. Even the total throughput increases, packet loss negatively affects the QoS of all flows.

We expected that if we collected NSI with shorter periods, we could get more accurate NSI. However, the accuracy of NSI depends on not only the collection period but also the size and density of topology. Thus, one needs to either tune the collection period based on the given network topology and traffic load, or propose new collection methods to increase the accuracy of NSI. As the future work, we plan to specifically study on how to eliminate the inconsistencies caused by periodically collected NSI. First, we plan to create some probabilistic link metrics that deal with inaccurate state information. Second, we plan to propose an effective NSI collection method that minimizes the load on the controller and the message overhead throughout the network.

REFERENCES

[1] D. Kreutz, F. Ramos, P. E. Veríssimo, 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] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined Internet architecture: Decoupling architecture from infrastructure," in *Proc. 11th ACM Workshop Hot Topics Netw.*, Oct. 2012, pp. 43–48.

[3] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Intelligent design enables architectural evolution," in *Proc. 10th ACM Workshop Hot Topics Netw.*, Nov. 2011, Art. no. 3.

[4] N. McKeown, "How SDN will shape networking," *Open Netw. Summit*, Oct. 2011.

[5] S. Shenker, M. Casado, T. Koponen, and N. McKeown, "The future of networking and the past of protocols, Oct. 2011," *Invited Talk Open Netw. Summit*, 2011.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[8] P. Goransson and C. Black, *Software Defined Networks: A Comprehensive Approach*. San Mateo, CA, USA: Morgan Kaufman, 2014.

[9] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

[10] RYU. (2017). *Component-Based Software Defined Networking Framework*. Accessed: Dec. 6, 2017.

[11] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw.*, Oct. 2010, Art. no. 19.

[12] M. Ramalho, "Intra- and inter-domain multicast routing protocols: A survey and taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 3, no. 1, pp. 2–25, 1st Quart., 2000.

[13] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 58–74, 3rd Quart., 2007.

[14] B. Wang and J. C. Hou, "Multicast routing and its QoS extension: Problems, algorithms, and protocols," *IEEE Netw.*, vol. 14, no. 1, pp. 22–36, Jan. 2000.

[15] C. Maihofer, "A survey of geocast routing protocols," *IEEE Commun. Surveys Tuts.*, vol. 6, no. 2, pp. 32–42, Dec. 2004.

[16] S. K. Singh, T. Das, and A. Jukan, "A survey on Internet multipath routing and provisioning," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2157–2175, 4th Quart., 2015.

[17] N. Chakchouk, "A survey on opportunistic routing in wireless communication networks," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2214–2241, Mar. 2015.

[18] D. Chen and P. K. Varshney, "A survey of void handling techniques for geographic routing in wireless networks," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 1, pp. 50–67, 1st Quart., 2007.

[19] F. Mansourkiaie and M. H. Ahmed, "Cooperative routing in wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 604–626, May 2015.

[20] Y. Tsado, K. A. A. Gamage, B. Adebisi, D. Lund, K. M. Rabie, and A. Ikpehai, "Improving the reliability of optimised link state routing in a smart grid neighbour area network based wireless mesh network using multiple metrics," *Energies*, vol. 10, no. 3, p. 287, 2017.

[21] J. Agarkhed, P. Y. Dattatraya, and S. R. Patil, "Performance evaluation of QoS-aware routing protocols in wireless sensor networks," in *Proc. 1st Int. Conf. Comput. Intell. Inform.* Springer, 2017, pp. 559–569.

[22] M. Z. Hasan, F. Al-Turjman, and H. Al-Rizzo, "Optimized multi-constrained quality-of-service multipath routing approach for multimedia sensor networks," *IEEE Sensors J.*, vol. 17, no. 7, pp. 2298–2309, Apr. 2017.

[23] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS routing algorithms for SDN: A comprehensive survey and performance evaluation," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 388–415, 1st Quart., 2018.

[24] X. Masip-Bruin, M. Yannuzzi, J. Domingo-Pascual, A. Fonte, M. Curado, E. Monteiro, F. Kuipers, P. Van Mieghem, S. Avallone, G. Ventre, P. Aranda-Gutiérrez, M. Hollick, R. Steinmetz, L. Iannone, and K. Salamatian, "Research challenges in QoS routing," *Comput. Commun.*, vol. 29, no. 5, pp. 563–581, 2006.

[25] S. Upadhyaya and G. Dhingra, "Exploring issues for QoS based routing algorithms," *Int. J. Comput. Sci. Eng.*, vol. 2, no. 5, p. 2010, 1792.

[26] R. G. Garroppo, S. Giordano, and L. Tavanti, "A survey on multi-constrained optimal path computation: Exact and approximate algorithms," *Comput. Netw.*, vol. 54, no. 17, pp. 3081–3107, 2010.

[27] E. Akin and T. Korkmaz, "Link-prioritized network state information collection in SDN," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[28] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.

[29] A. Orda, S. Kama, D. Williams, R. Guerin, T. Przygienda, and G. Apostolopoulos, *QoS Routing Mechanisms and OSPF Extensions*, document RFC 2676, 1999.

[30] M. Kodialam and T. V. Lakshman, "Minimum interference routing with applications to MPLS traffic engineering," in *Proc. 19th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Mar. 2000, pp. 884–893.

[31] E. Crawley, H. Sandick, R. Nair, and B. Rajagopalan, *A Framework for QoS-Based Routing in the Internet*, document RFC 2386, 1998.

[32] A. B. Bagula, M. Botha, and A. E. Krzesinski, "Online traffic engineering: The least interference optimization algorithm," in *Proc. IEEE Int. Conf. Commun.*, vol. 2, Jun. 2004, pp. 1232–1236.

[33] I. Olszewski, "The improved least interference routing algorithm," in *Image Processing and Communications Challenges*. Springer, 2010, pp. 425–433.

[34] M. Aslan and A. Matrawy, "On the impact of network state collection on the performance of SDN applications," *IEEE Commun. Lett.*, vol. 20, no. 1, pp. 5–8, Jan. 2016.

[35] Z. Guo, M. Su, Y. Xu, Z. Duan, L. Wang, S. Hui, and H. J. Chao, "Improving the performance of load balancing in software-defined networks through load variance-based synchronization," *Comput. Netw.*, vol. 68, pp. 95–109, Aug. 2014.

[36] E. Akin and T. Korkmaz, "Routing algorithm for multiple unsplittable flows between two cloud sites with QoS guarantees," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2017, pp. 917–923.

[37] K. Nahrstedt, B. Yu, J. Liang, and Y. Cui, "Hourglass multimedia content and service composition framework for smart room environments," *Pervasive Mobile Comput.*, vol. 1, no. 1, pp. 43–75, 2005.

[38] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient SDN control plane," in *Proc. 8th Int. Workshop Resilient Netw. Design Modeling (RNDM)*, Sep. 2016, pp. 253–259.

[39] J. G. V. Pena and W. E. Yu, "Development of a distributed firewall using software defined networking technology," in *Proc. 4th IEEE Int. Conf. Inf. Sci. Technol.*, Apr. 2014, pp. 449–452.

[40] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[41] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

[42] L. Ochoa-Aday, C. Cervelló-Pastor, and A. Fernández-Fernández, "eTDP: Enhanced topology discovery protocol for software-defined networks," *IEEE Access*, vol. 7, pp. 23471–23487, 2019.

[43] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan, "Multi-controller based software-defined networking: A survey," *IEEE Access*, vol. 6, pp. 15980–15996, 2018.

[44] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, S. Shenker, and T. Hama, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, Oct. 2010, pp. 1–6.

[45] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for OpenFlow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.

[46] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, Aug. 2012, pp. 1–6.

[47] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, Sep. 2012.

[48] Y.-N. Hu, W.-D. Wang, X.-Y. Gong, X.-R. Que, and S.-D. Cheng, "On the placement of controllers in software-defined networks," *J. China Univ. Posts Telecommun.*, vol. 19, pp. 92–171, Oct. 2012.

[49] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. 3rd workshop Hot Topics Softw. Defined Netw.*, Aug. 2014, pp. 31–36.

[50] D. Tuncer, M. Charalambides, S. Clayman, and G. Pavlou, "On the placement of management and control functionality in software defined networks," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, Nov. 2015, pp. 360–365.

[51] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2013, pp. 18–25.

[52] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Comput. Commun.*, vol. 67, pp. 1–10, Aug. 2015.

[53] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. 25th Int. Teletraffic Congr. (ITC)*, Sep. 2013, pp. 1–9.

[54] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "POCO-PLC: Enabling dynamic Pareto-optimal resilient controller placement in SDN networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr./May 2014, pp. 115–116.

[55] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[56] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.

[57] *OpenFlow: Proactive vs Reactive*. Accessed: Jan. 23, 2018. [Online]. Available: http://networkstatic.net/openflow-proactive-vs-reactive-flows/
[58] *Reactive, Proactive, Predictive: SDN Models*. Accessed: Jan. 23, 2018. [Online]. Available: https://devcentral.f5.com/articles/reactive-proactive-predictive-sdn-models
[59] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A hybrid hierarchical control plane for flow-based large-scale software-defined networks," *IEEE Trans. Netw. Service Manage.*, vol. 12, no. 2, pp. 117–131, Jun. 2015.
[60] S. Chaudhuri, S. R. Das, H. S. Paul, and S. Tirthapura, "Distributed computing and networking," in *Proc. Int. Conf. Distrib. Comput. Netw.*, 2013.
[61] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 351–362, Oct. 2010.
[62] C. Trois, M. D. Del Fabro, L. C. E. de Bona, and M. Martinello, "A survey on SDN programming languages: Toward a taxonomy," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2687–2712, 4th Quart., 2016.
[63] A. Goldberg and R. Tarjan, "Solving minimum-cost flow problems by successive approximation," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 7–18.
[64] J. Moy, *OSPF Version 2*, document RFC 2178, Jul. 1997.
[65] T. Korkmaz and M. Krunz, "Bandwidth-delay constrained path selection under inaccurate state information," *IEEE/ACM Trans. Netw.*, vol. 11, no. 3, pp. 384–398, Jun. 2003.
[66] M.-C. Yuen, W. Jia, and C.-C. Cheung, "Simple mathematical modeling of efficient path selection for QoS routing in load balancing," in *Proc. IEEE Int. Conf. Multimedia Expo*, vol. 1, Jun. 2004, pp. 217–220.
[67] M. R. Garey and D. S. Johnson, *A Guide to the Theory of NP-Completeness*, vol. 70. New York, NY, USA: Freeman, 1979.
[68] S. Chen and K. Nahrstedt, "An overview of quality of service routing for next-generation high-speed networks: Problems and solutions," *IEEE Netw.*, vol. 12, no. 6, pp. 64–79, Nov./Dec. 1998.
[69] G. B. Figueiredo, N. L. S. Da Fonseca, and J. A. S. Monteiro, "A minimum interference routing algorithm with reduced computational complexity," *Comput. Netw.*, vol. 50, no. 11, pp. 1710–1732, Aug. 2006.
[70] E. Akin and T. Korkmaz, "An efficient binary-search based heuristic for extended unsplittable flow problem," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2017, pp. 831–836.
[71] H. Pham and B. Lavery, "New dynamic link weight algorithm for on-line calculation of LSPS in MPLS networks," in *Proc. 8th Int. Conf. Commun. Syst. (ICCS)*, vol. 1, Nov. 2002, pp. 117–121.
[72] W. Sa-Ngiamsk, S. Thipchaksurat, and R. Varakutsiripunth, "A bandwidth-based constraint routing algorithm for multi-protocol label switching networks," in *Proc. 6th Int. Conf. Adv. Commun. Technol.*, vol. 2, Feb. 2004, pp. 933–937.
[73] F. Ricciato and U. Monaco, "On-line routing of MPLS tunnels with time-varying bandwidth profiles," in *Proc. Workshop High Perform. Switching Routing (HPSR)*, Apr. 2004, pp. 321–325.
[74] M. Zhu, W. Ye, and S. Feng, "A new dynamic routing algorithm based on minimum interference in MPLS networks," in *Proc. 4th Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Oct. 2008, pp. 1–4.
[75] B. Wang, X. Su, and C. L. P. Chen, "A new bandwidth guaranteed routing algorithm for MPLS traffic engineering," in *Proc. IEEE Int. Conf. Commun. (ICC)*, vol. 2, Apr./May 2002, pp. 1001–1005.
[76] A. Kotti, R. Hamza, and K. Bouleimen, "Bandwidth constrained routing algorithm for MPLS traffic engineering," in *Proc. Int. Conf. Netw. Services (ICNS)*, Jun. 2007, p. 20.
[77] P. Krachodnok, "Constraint-based routing with maximize residual bandwidth and link capacity—Minimize total flows routing algorithm for MPLS networks," in *Proc. 5th Int. Conf. Inf. Commun. Signal Process.*, Dec. 2005, pp. 1509–1514.
[78] *Mininet*. Accessed: Feb. 18, 2018. [Online]. Available: http://mininet.org
[79] *iPerf—The TCP, UDP and SCTP Network Bandwidth Measurement Tool*. Accessed: Jan. 25, 2018. [Online]. Available: https://iperf.fr/
[80] D. E. Comer, *Internetworking con TCP/IP*, vol. 1. London, U.K.: Pearson, 2006.
[81] *Restrictions to Mininet, Network Simulation With Mininet*. Accessed: Feb. 26, 2019. [Online]. Available: https://projectmincpm.wordpress.com/2015/06/22/restrictions-to-mininet/
[82] A. Azzouni, N. T. M. Trang, R. Boutaba, and G. Pujolle, "Limitations of OpenFlow topology discovery protocol," in *Proc. 16th Annu. Medit. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, Jun. 2017, pp. 1–3.

ERDAL AKIN was born in Ceyhan, Adana, Turkey. He graduated from the Department of Mathematics, Yildiz Technical University, Istanbul, in 2008. He received the master's and Ph.D. degrees from the Department of Computer Science, The University of Texas at San Antonio (UTSA), in Spring 2014 and December 2018, respectively. He completed his prerequest courses, in 2012. He worked with Dr. Turgay Korkmaz and focused on routing algorithms and network state information collection methods in software defined networking. In 2009, he was a recipient of the Republic of Turkey Ministry of National Education scholarship to study in USA.

TURGAY KORKMAZ received the B.Sc. degree (Hons.) in computer science and engineering from Hacettepe University, Ankara, Turkey, in 1994, the M.Sc. degree in computer engineering from Bilkent University, Ankara, the M.Sc. degree in computer and information science from Syracuse University, Syracuse, NY, USA, in 1996 and 1997, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Arizona, in December 2001, under the supervision of Dr. M. Krunz. In January 2002, he joined The University of Texas at San Antonio, as an Assistant Professor with the Computer Science Department. He is currently an Associate Professor with the Computer Science Department. He is also involved in the area of computer networks, network security, network measurement and modeling, and Internet related technologies. His research interests include quality-of-services (QoS)-based networking issues in both wireline and wireless networks. He received his tenure, in September 2008.