DOLPHIN: Dynamically Optimized and Load Balanced Path for Inter-Domain SDN Communication

Zohaib Latif, Kashif Sharif, *Senior Member, IEEE*, Fan Li, *Member, IEEE*, Md Monjurul Karim, Sujit Biswas, *Member, IEEE*, Madiha Shahzad, *Member, IEEE*, and Saraju P. Mohanty, *Senior Member, IEEE*

***Abstract*—Software-Defined Networking has become an integral technology for large scale networks that require dynamic flow management. It separates the control function from data plane devices and centralizes it in a domain controller. However, only a limited number of switches can be managed by a single and centralized controller which introduces challenges such as scalability, reliability, and availability. Distributed controller architecture resolves these issues but also introduces new challenges of uneven load and traffic management across domains. As real-world networks have redundant links, hence a significant challenge is to distribute traffic flows on multiple paths, within a domain, and across multiple independent domains. The selection of ingress and egress switches becomes even more problematic if the intermediate domain is non-cooperative. In this work, we propose a Dynamically Optimized and Load-balanced Path for Inter-domain (DOLPHIN) communication system, a customized solution for different SDN controllers. It provides control beyond the virtual switch elements in intra and inter-domain communication and extends the range of programmability to wireless devices, such as the Internet of Things or vehicular networks. Extensive simulation results show that the traffic load is distributed evenly on multiple links connecting different domains. We model data center communication and 5G vehicular network communication to show that, by load balancing the flow completion times of the different types of network traffic can be significantly improved.**

Manuscript received May 31, 2020; revised October 28, 2020; accepted December 13, 2020. Date of publication December 18, 2020; date of current version March 11, 2021. The work of F. Li is supported by the National Natural Science Foundation of China No. 62072040, 61772077 and the Beijing Natural Science Foundation No. 4192051. The associate editor coordinating the review of this article and approving it for publication was M. F. Zhani. *(Corresponding authors: Kashif Sharif; Fan Li.)*

Zohaib Latif is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China, and also with the Department of Computing, Riphah International University, Faislabad 44000, Pakistan (e-mail: z.latif@bit.edu.cn).

Kashif Sharif and Fan Li are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China, and also with Beijing Engineering Research Center of High Volume Language Information Processing and Cloud Computing Applications, Beijing Institute of Technology, Beijing 100081, China (e-mail: kashif@bit.edu.cn; fli@bit.edu.cn).

Md Monjurul Karim is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China (e-mail: mkarim@bit.edu.cn).

Sujit Biswas is with the Faridpur Engineering College, University of Dhaka, Dhaka 1000, Bangladesh (e-mail: sujitedu@bit.edu.cn).

Madiha Shahzad is with the School of Sciences, University of Central Lancashire, Pyla 7080, Cyprus (e-mail: mshahzad1@uclan.ac.uk).

Saraju P. Mohanty is with the Department of Computer Science and Engineering, University of North Texas, Denton, TX 76207 USA (e-mail: saraju.mohanty@unt.edu).

Digital Object Identifier 10.1109/TNSM.2020.3045725

***Index Terms*—Software defined networking, inter-domain communication, vertical programmability, load balancing.**

I. Introduction

In TRADITIONAL networks, every network layer element has its own decision making power and control mechanisms. With the rapid growth of the Internet and a large number of autonomous systems, it has become complicated for network administrators to configure each device separately [1]. Software-Defined Networking (SDN) [2], [3] is a networking paradigm that decouples control logic from the data plane and centralizes it as a software-based entity. Due to this separation, data plane devices become simple forwarding nodes and update the control plane by using well-defined Application Programmable Interfaces (APIs) [4]. OpenFlow [5] is one of the most widely used southbound API used for this purpose. Based on the information provided by data plane devices, the control plane generates a global view of the network and pushes forwarding rules to the data plane. The high-level management plane communicates with the control plane to enforce various policies to control the network. Due to its promising features and flexible network management, SDN has been deployed in various network environments, such as data centers [6] and wide-area networks [7].

Once the routing control is shifted to the centralized controller (as compared to independent routers), hence the traditional switches have to be replaced with SDN switches, which use flow tables for forwarding. As soon as the switch receives a new packet, it matches the packet header with flow entries present in the flow table. In response to this matching (if it finds any flow entry), the packet is forwarded to the corresponding port. Otherwise, the packet is either dropped or forwarded to the controller using the Packet_IN message. In response to this request, the controller generates a Packet_OUT message and installs flow entry on the switches along the path. Eventually, the switch takes action against this entry and forwards the packet to the port for data transfer. Once a flow entry is installed, the switch takes action according to this rule. In the case of multiple paths between source and destination, the controller picks one path to install (or selects an existing flow group), however, this path remains the same for the duration of that flow. This may create an overload on specific network links, while other possible paths remain underutilized.

The architecture of SDN is centralized, and only a limited number of switches can be managed by a single controller [8]. Moreover, whenever a new packet arrives at a data plane element, it requires the involvement of a controller to process this packet. An overloaded controller cannot process this packet immediately, which may cause additional delay [9]. The centralized architecture also introduces reliability issues due to a risk of controller failure [10]. To address the above issues, distributed controller architecture is used which can be classified as *distributed (flat)* and *hierarchal* architectures where every controller is responsible for its domain and updates either neighboring controllers or a root controller [11], [12]. Similar to the centralized domain, the issue of uneven load also exists in multiple domains but the extent is far greater as the different domains are controlled by their respective controllers. Inter-controller coordination becomes a complex task, as a path selected by one domain controller may not be the optimal path for the other domain.

In real-world networks, there are always multiple paths between source and destination pairs. These pairs may either be in the same domain or different domains. Load balancing is a method that is used to distribute the traffic load evenly on different paths. Major goals of load balancing are to: maximize throughput, minimize delay, traffic shaping, and improved flow completion time. Load balancing can be divided into two categories as static and dynamic. In static load balancing, the route is calculated and allocated before the traffic transmission, and cannot be changed during data transmission. Since the behavior of the user cannot be predicted, so it has poor flow scheduling and causes congestion if the source transmits a large number of flows. Dynamic load balancing, on the other hand, can schedule traffic according to traffic statistics which is updated periodically and provides better results as compared to static load balancing. However, this can create overhead in flow installation and processing.

In some specialized applications such as the Internet of Things (IoT) [13], the network is composed of heterogeneous devices like, home appliances, sensors, and other electronic devices that can transfer data freely through the Internet. These devices are connected with data plane elements (i.e., the SDN switches), however as they are not Openflow compatible, hence their position in the layered SDN structure is not well defined [14]. In this work, we consider these devices as part of the perception plane, as shown in Figure 1. In the global view of the controller, perception plane elements cannot be seen and OpenFlow cannot install flow rules on these elements. Hence, load balancing and global view for the elements of perception plane is an added challenge in IoT domains.

To solve the problems of the uneven load over different paths between source and destination residing in multiple independent domains, and to provide flow control over perception plane elements for optimized traffic flow, we propose Dynamically Optimized and Load-balanced PatH for INter-domain SDN Communication (DOLPHIN). The proposed solution provides dynamic load balancing where paths can be changed on the fly during data transfer. The solution is implemented as a module on the SDN controller and provides horizontal load balancing (inter-domain) and vertical extension (into perception plane). In horizontal load balancing, it enables dynamic traffic load optimization over multiple paths in a single as well as in multiple domains. Moreover, it manages traffic load among multiple domains even if they are not directly connected and non-cooperative domain(s) are present between the source and destination. In the vertical extension, it provides control over elements that are in the perception plane and balances load accordingly. The main contributions of this work are as follows.

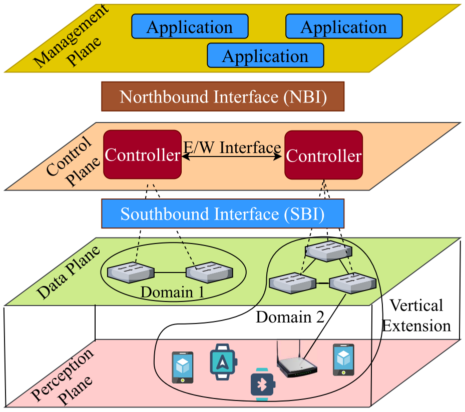


Fig. 1. Architecture of SDN layers with perception plane as the vertical extension of data plane.

- We propose a complete path computation and topology extension (horizontal and vertical) system in a multi-domain SDN environment.
- The proposed solution balances traffic load evenly on multiple links, not only in single but across multiple domains. It computes the weight value of each path based on multiple metrics to find the optimal path at any given time.
- In the case of a non-cooperative domain between source and destination, it selects egress and ingress gateway nodes that provide an optimal path across that domain, while balancing the load in cooperative domains.
- The solution creates sub-controllers on access nodes to extend the reach of SDN into the perception plane and is capable of installing flows on mobile devices, with minor application layer modifications.
- Extensive evaluations have been done to show the proof of concept, and implementation has been done in a data center and vehicular network environment to determine the performance.

The rest of this article is organized as; Section II describes the background of various existing algorithms, while system design and architecture is presented in Section III. Section IV provides details of horizontal intra-domain load balancing whereas, inter-domain load balancing and vertical extension are discussed in Section V. Section VI presents the simulation setup for experiments and performance evaluation. Conclusion and future work are discussed in Section VII.

TABLE I
COMPARATIVE ANALYSIS OF EXISTING SOLUTIONS

Ref.	Intra-Domain	Inter-Domain	Vertical Extension
[18]	Controller pool usage	-	No
[20]	Flow load balancing	No	No
[21]	Path optimization	No	No
[22]	Multipath selection		No
[23]	Multipath assignment	No	No
[24]	Flow load balancing	No	No
[25]	Dynamic load balancing	No	No
[26]	Dynamic, Threshold-based, Congestion avoidance	No	No
[27]	-	P2P SDN, Loop avoidance	No
[28]	-	Scalability using coordinator controller	No
[29]	Controller and Link load balancing	No	No
[30]	-	Switch migration	No
[31]	-	Switch migration, Controller hierarchy	No
[32]		Inter-Controller msgs. reduction	No
[33]	ML for controller load balancing	-	No
[34]	Path optimization	-	For UE only
[35]	MPLS for flow optimization	-	No
[36]	Flow path optimization	No	No
[37]	Multipath Solution	No	No
This Work	Dynamic flow management	Dynamic flows (w/out cooperative domain)	Yes (with multipath)

II. RELATED WORKS

Initially, SDN was based on a single controller design with a global network view responsible for managing the complete network. Controllers like NOX [15] and POX [16] are primary examples of such implementation. However, due to the scalability and performance limitations, several studies suggest the use of distributed controllers, such as [10], [17]–[19]. These solutions enabled SDN for large scale networks but introduced some new challenges in terms of load balancing and dynamic optimal path selection. Load balancing on different paths and load distribution of the controller are two different challenges. In this work, we address the challenge of dynamic load balancing on different paths among source and destination, in traditional networks and perception plane.

A. Load Balancing for Traffic Management

The work in [20] aims at dynamic balancing of traffic load in SDN for flows under a single domain. It selects a transitory path for the fresh flows, and if the load on a link is unstable, it diverts the flows onto a different path. The main issue with it is that it sometimes changes the paths without need which creates oscillation. In [21] authors discussed the shortest path, shortest feasible path, and widest path by using modified Dijkstra's algorithm. However, this solution is also aimed at a single domain SDN. In [22] authors described a multipath routing scheme where the best optimal path is selected from multiple paths under a single controller. The authors used the Ryu controller to measure the network parameters for quality of service such as latency, packet loss, and throughput. Authors in [23] used the POX controller and enhanced network performance by using a proactive approach where different paths are assigned to multiple flows evenly based on the bandwidth of these paths. Authors in [24] provided load balancing in the Data Center Network environment using the POX controller and assessed its performance through OpenFlow Statistics. Their technique also implemented path restoration and traffic classification. DLPO [25] is another solution for dynamic load balancing under a single domain where flows of a congested path are redirected to a lightly loaded path after updating the flow tables of associated switches to reduce the risk of packet loss. mED-SDN [26] describes an approach that uses REST API and modified Dijkstra's algorithm for SDN. It uses a threshold value, which is set by the algorithm for congestion prevention mechanism. It finds a new path when the bandwidth exceeds the threshold value. Inter-domain multipath routing is addressed by using traffic engineering in a P2P SDN [27], where the authors discussed information exchange among multiple domains, path aggregation, and overhead. The authors focused on routing loop formation and proposed a solution using a routing table, topology table, and an advertisement table. The work in [28] proposed a multi-domain architecture and used a coordinator controller, which is connected with domain controllers using a unified NBI. However, this work does not address load balancing on multiple paths among domains. Its primary focus is the scalability issue for multiple domains handled by controllers from different vendors.

B. Balancing of Controller Load

Load balanced routing for links and controller (LBR-LC) [29] exploits a routing based algorithm and analyzes its approximation performance. It reduces the response time of the controller (by load bounds) and achieves link load balancing (using network area bounds). The work in [30] proposed a method for balancing the uneven load in the control plane by shifting load away from heavily loaded controllers. This is achieved through switch migration. However, the migration process requires some delay during the re-configuration of devices, especially in large scale networks. Another solution [31] works as a controller module, and switches are connected with multiple controllers but only one controller is connected as the master. Each controller gathers information about its domain and compares it to that of other domains. If its load is greater as compared to the rest of the controllers, then it selects and migrates switches to a less loaded controller. However, this requires extensive information about the load and synchronization of controllers which may reduce overall performance. In [32] authors proposed a load balancer application to reduce the inter controller control messages. At the same time, the authors proposed a multi-policy controller where multiple applications can control the network. As the switch to controller mapping is static, hence a controller may become overloaded if a large number of flows arrive. Elasticon [18] presents an architecture where the load of a controller is computed, and the controller pool can be dynamically expanded and shrunk, which enhances the network performance and throughput. In [33], the authors proposed multi-agent reinforcement learning to balance the controller load. It uses offline training and online decision making to balance the workload of controllers.

C. Path Optimizations

In [34] authors propose full, partial, and heuristic path computation optimized strategies specifically for 5G enabled SDN-based transport network (wireless and wired), emphasizing increased QoS experience. All the proposed schemes use linear integer programming optimization modules to recompute all the available paths between the OVS switches, SDN controller, servers, and UEs to determine the network's optimal configuration. There is no implementation of these algorithms, although the paper mentions ONOS. Traditional MPLS labeling has been applied in [35] to improve the network resource utilization by reducing the number of flow entries while reducing the controller-switch communication overhead. Flow tagging has been introduced into modified OpenFlow switches to achieve flow-based forwarding aggregation and multipath communication simultaneously. A flow-rule placement solution is proposed in [36] to incorporate the maximum number of flows while locating an immediate end-to-end path. Despite showing promising results, the solution does not consider using a distributed controller while existed flow-rules likely to reappear repeatedly in the controller. Path selection for egress traffic in the stub network is presented in [37]. The solution calculates the available path's capacity and disseminates the traffic towards multiple paths using passive measurement techniques. A self-developed discrete-event simulator is used to form a client-server scenario that captures traffic in two identical data producer networks. The performance has been evaluated against the number of flows, their sizes, and their arrival time.

Contrary to the above-discussed studies, in this work, we focus on communication in multi-domain environments, with emphasis on path load balancing, selection of egress/ingress switches, and vertical extension to the data plane. The principle of path load balancing is weight-based and can be applied in both intra-domain and inter-domain communication. Consideration to non-cooperative domains is also given when multiple domains are involved. A major function of this work is to extend the data plane to the perception plane, where delegated control functionality is placed on access devices for multi-path multi-hop communication in a wireless environment. Recently, in [38] the authors have listed all of these objectives as major future directions of load-balanced SDN systems.

III. System Architecture

This section presents the overall architecture for the proposed approach. The architecture can be divided into two parts, the architecture of SDN planes, and the system model. In the architecture of SDN planes, we elaborate on the functionalities of different planes, while in the system model we describe the overall multi-domain design and different notations used in this work. The system model is further classified into two parts; horizontal load balancing and vertical extension. In horizontal load balancing, the load is distributed evenly on different links of data plane elements whereas, load balancing between data plane elements and perception plane elements is discussed in vertical extension.

A. Architecture of Planes

The architecture of Dolphin is aimed at large scale SDN networks. Here, we first explain the architecture and key components, as shown in Figure 1. It shows the high-level architecture for both horizontal load balancing and the vertical extension into the perception plane.

Perception Plane: At the base of the architecture lies the perception plane, which has different end-devices and some of these devices are connected with data plane elements by means of different interfaces (e.g., WiFi). Traditionally, the SDN architecture has SDN switches (capable of understanding Openflow) in the data plane and disregards other end-hosts. However, with the increase in diversity of end hosts and a complete topological structure, the extension of the data plane is referred to as the perception plane. The devices in the perception plane include mobile phones, sensors, vehicles, UAVs, or any other data generation/reception point, and may be connected to each other via different interfaces (e.g., WiFi, BlueTooth, NFC, ZigBee, and 5G). Nodes connected with data plane elements (SDN Switches or vSwitches) can be detected by the SDN controller (through SBI); however, end devices that are connected with these nodes are beyond SDN Switch elements and cannot be accessed by the controller. This plane involves the vertical extension of Dolphin, however, in horizontal load balancing, only data plane devices are used. The proposed vertical extension in this work allows access and flow installation at the perception level.

Data Plane: The data plane is above the perception plane, which has multiple domains. Each domain has SDN enabled (hardware or software) switches acting as forwarding nodes and are managed by a controller (which is part of the control plane). Connectivity among these switches enables multiple paths among different source and destination pairs. Moreover, border gateway switches of different domains are connected to each other via multiple links. Network information, link stats, and flow table updates are exchanged between controller and switches via the southbound interface (i.e., OpenFlow protocol).

Control Plane: The control plane in the proposed architecture contains several distributed SDN controllers (potentially from different vendors), which may or may not be connected in some hierarchy. Controllers manage flow tables of SDN switches under their respective domains. The SDN controller features represent the base network functions that are executed by the physical and virtual network managers of each controller. These functions extract topology updates, link states, and statistics (e.g., latency, packet loss ratio, and link utilization). After collecting this information, the controller can forward it to management plane applications.

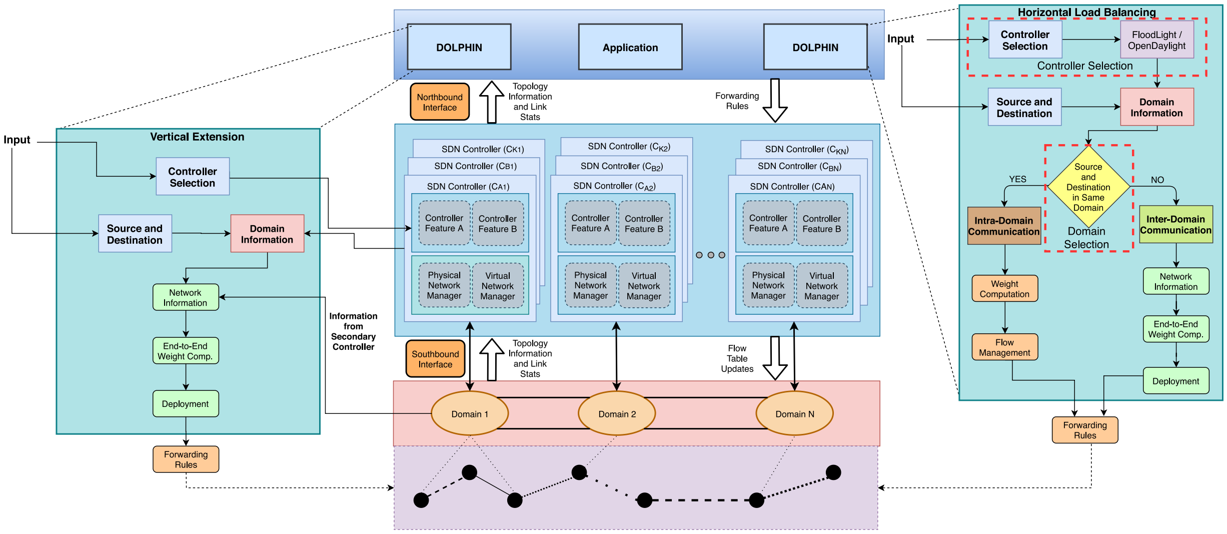


Fig. 2. Overall system design and modules.

Management Plane: The proposed Dolphin core algorithms can work as a management application in the management plane, as well as an integral part of the controller. It is preferable to use it as a management application, as this allows the control plane to have multi-vendor SDN solutions. The objective is to balance the load horizontally and vertically across the entire data and perception planes.

B. System Design and Communication Models

Horizontal load balancing is done entirely in the data plane among different domains, while vertical balancing additionally involves the perception plane. Figure 2 shows the overall design and the working principle, along with different modules for the horizontal and vertical extension. In this subsection, we describe the connectivity structure of the proposed solution and different notations used in this work.

1) Horizontal Model: Horizontal load balancing is classified into two parts: intra-domain and inter-domain. In intra-domain balancing, the source and destination belong to the same domain, whereas, if source and destination are in different domains then inter-domain balancing is done. The system model for horizontal load balancing in the proposed system can be observed from the sample topology as shown in Figure 3. The inter-domain network is shown at the top as a collection of different domains connected through multiple links. The magnified view of each domain shows the control and data plane of each. We can observe that S1-S4 are SDN switches, which connect different end machines labeled as H1-H3. Also, this domain has SDN switches acting as gateways to other domains (GW1-GW3). Assume that the network has $D = \{d_1, d_2, d_3 \cdots d_n\}$ set of SDN domains, then the topology graph can be represented as G $= (V^d,$E), where $V^{d_i} = \{v_1^{d_i}, v_2^{d_i}, v_3^{d_i} \cdots v_m^{d_i}\}$ represents switches of domain d_i, and E $= \{(u, v) :\ u, v \in V\}$ is a set of edges to connect V switches. Any path between a source s and destination t switch in a domain d is represented as $p_i = \{v_s^d, v_1^d, v_2^d \cdots v_t^d\}$ and $p_i \in$ P where P represents a set of all shortest paths between source and destination pairs. It should be noted that traffic originator is a host, however, from a controller's perspective, it is switch from where the flow begins. Hence, we have used switches to represent source and destination. Moreover, every edge E has a weight W_j, where $j \in [1 - \text{M}]$ is a series of M non-negative weights and cost functions which includes; link weight, latency, and packet loss ratio to ensure the Quality of Service (QoS). Finally, the total weight of path p_i can be calculated as $W_{p_i} = \sum_{e=0}^{n} W_e$, where W_e is weight of each link in path p_i. Figure 3 also depicts that some domains may be connected directly (connecting through GW switches), while others have to send traffic across other domains to reach a particular destination. Most of these domains are connected via multiple paths. For example, D1 is connected with D2 and D3 through multiple links.

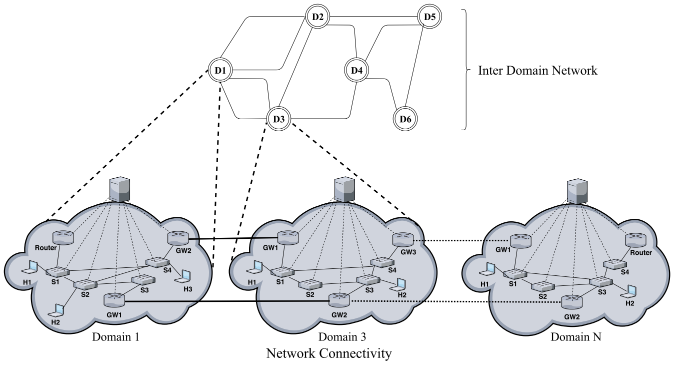


Fig. 3. Topological structure used for the system model.

Formally we can define the problem as: a source $v_s^{d_i}$ in domain d_i intends to communicate with $v_t^{d_j}$ in domain d_j, where $v_s^{d_i}$, $v_t^{d_j} \in V^d$ and path p_i is said to be best optimal path if: $W_{p_i} \leq W_{p_j}$ where $p_i, p_j \in$P$\wedge$ $p_i \neq p_j$.

Intra-Domain Handling: The main objective of intra-domain load balancing is to balance the traffic evenly on different paths when source and destination belong to the same domain. For example, let $v_s^{d_j}$ and $v_t^{d_j}$ are source and destination switches of domain d_j and there are two possible routes among this pair, which can be represented as $v_s^{d_j} \rightarrow v_a^{d_j} \rightarrow v_t^{d_j}$ and $v_s^{d_j} \rightarrow v_b^{d_j} \rightarrow v_t^{d_j}$. To find and redirect traffic to a less loaded path, a dynamic and adequate solution is required which is robust in determining the network state and adjusting the flow path to optimize the load. Note that this work's objective is to change paths while flows are in progress.

Inter-Domain Handling: When the source and destination switches are in different domains (and under different controllers), two scenarios can be realized as directly connected domains or indirectly connected domains.

- *Directly Connected:* In this scenario, there are two or more domains (with their independent controllers) that are either adjacent to each other or are cooperative domains (i.e., under the same administration). Continuing from previous scenario, the destination switch is now present in a different domain d_k. Hence, the available paths between source destination are $v_s^{d_j} \rightarrow v_a^{d_j} \rightarrow v_a^{d_k} \rightarrow v_t^{d_k}$ and $v_s^{d_j} \rightarrow v_b^{d_j} \rightarrow v_b^{d_k} \rightarrow v_t^{d_k}$ where $v_a^{d_j}$, $v_b^{d_j}$ are the gateway switches for domain d_j and $v_a^{d_k}$, $v_b^{d_k}$ are the gateway switches for domain d_k. It is possible that for switch $v_s^{d_j}$ best optimal path is through $v_a^{d_j}$ whereas, for domain k optimal path for desired destination is via switch $v_b^{d_k}$. This creates a challenging situation as one of the domain will have to use a sub-optimal path for communication. It is worth mentioning here that in SDN, a single switch can be connected to multiple domain controllers. If it is, then, one controller becomes master and the other is a slave for that switch. Only the master controller can read and write flow rules, whereas the slave controller can only read the information to generate a global view. Hence, a gateway switch always belongs to a single domain (where the domain is defined by the master controller).
- *Indirectly Connected Domains:* In real-world networks, probability of intermediate domains between any source and destination domains is very high. Hence, when the source switch $v_s^{d_j}$ and destination switch $v_t^{d_n}$, have to communicate over domains where flow installation is not possible, then only best-effort delivery can be guaranteed. For example, domain d_m is a non co-operative domain and is not willing to share its topological information thus; the available paths between source destination are $v_s^{d_j} \rightarrow v_a^{d_j} \rightarrow \mathrm{X} \rightarrow v_a^{d_n} \rightarrow v_t^{d_n}$ and $v_s^{d_j} \rightarrow v_b^{d_j} \rightarrow \mathrm{X} \rightarrow v_b^{d_n} \rightarrow v_t^{d_n}$ where $v_a^{d_j}$, $v_b^{d_j}$ are the gateway switches for domain d_j and $v_a^{d_n}$, $v_b^{d_n}$ are the gateway switches for domain d_n.

2) Vertical Modeling: In order to provide the vertical load balancing, end devices connected with perception plane elements must be visible in the global topology at the controller. In practice, these end devices cannot be observed or accessed by the controller because OpenFlow works only at the SDN switch level. Our first goal is to generate the global network view along with end devices connected in perception plane elements. After creating the global view, the flows should be created which optimally connect the end devices, and then the load must be balanced evenly on multiple paths among them.

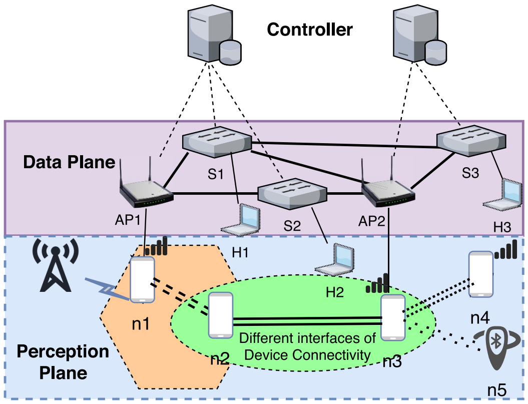


Fig. 4. Unified structure for the vertical extension.

Figure 4 shows two SDN domains where the perception plane is at the bottom where $n1$ and $n3$ are the perception plane elements. These elements can be further connected to different end devices (e.g., n2, n4, and n5) via different short-range radio interfaces (Bluetooth, NFC, DSRC, etc.). Moreover, these end devices could have multi-hop routing when they cannot communicate with access points directly. In this work, we provide the access points with similar capabilities as SDN switches, which are connected to the other wired-domain elements. In a communication scenario, assume that n1 and n3 want to transfer data, then in traditional SDN there could be different paths available, i.e., n1 → AP1 → S2 → AP2 → n3 and n1 → AP1 → S1 → AP2 → n3. However, in this work we extend the reach of the controller to the perception plane thus, making it capable of installing flows within the wireless domain. With the knowledge of n2's existence, a more optimal path can be established as n1 → n2 → n3. A practical example of such systems can be found in vehicular networks, where the same vehicle can be accessed through different roadside units (RSU) and vehicle-to-vehicle (V2V) communication.

IV. Horizontal Intra-Domain Load Balancing

In this section, we discuss load balancing among data plane elements inside the control of a single domain. Each source and destination pair is connected via multiple paths, as shown in the example topology of Figure 5. Switches S1 and S4 are connected through multiple paths as; S1→S2→S4 and S1→S3→S4. The proposed Dolphin intra-domain system runs in the management plane and redirects traffic on the less loaded path by re-configuring the flow table dynamically. It is important to clarify here that, while other state-of-the-art solutions install the flow for its lifetime, the proposed solutions dynamically changes the path, if it can find a better one. There are various sub-modules of the Dolphin system are discussed below and the complete process is given in Algorithm 1 for least weighted path selection.

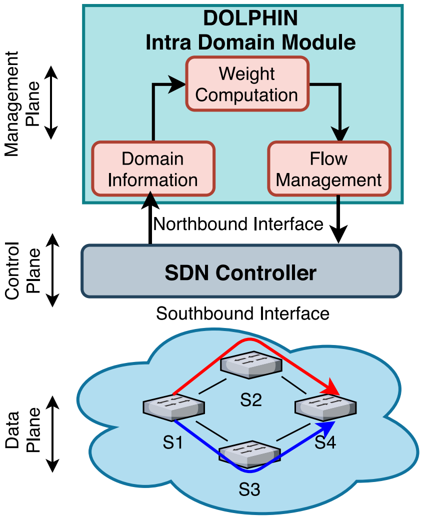


Fig. 5. Intra-domain system and its sub-modules.

Algorithm 1 Intra-Domain Path Selection Process

```
procedure INTRA–DOMAIN
    G = ($V^{d_i}$,E)
    $V^{d_i} = \{v_1^{d_i}, v_2^{d_i}, v_3^{d_i}, \dots v_n^{d_i}\}$ where $v_s^{d_i}$, $v_t^{d_i} \in V^{d_i}$ in $d_i$
    $E$: a set of edges from $v_s^{d_i}$ to $v_t^{d_i}$ in Graph $G$
    $P_i$: a set of non-disjoint paths from $v_s^{d_i}$ to $v_t^{d_i}$
    $W_{P_i}$: a set of weight of all paths
    $P_b$: best Path
    F: a set of switches to install flows
    $P_i \leftarrow$ G
    for each path $p \in P_i$ do
        $W_{P_i} = \sum_{e=0}^{n} W_e$
    end for
    $P_b = min(P_i, W_{p_i})$
    $F \longleftarrow$ list of switches in $P_b$
    return $F$
end procedure
```

Domain Information Sub-Module: The main objective of this sub-module is to generate the graph G = (V^d, E) which reflects the topology of the entire domain. It obtains the necessary information of network devices (e.g., IP & MAC addresses) and links (e.g., switch & port connectivity) from the controller. It also takes a record of gateway switches connected to its neighboring domains, which helps in inter-domain communication. The current state of network elements is collected in JSON and XML format using API interfaces of respective controllers. Moreover, this module retrieves port statistics from each switch of the network which helps to find the current value of link utilization. Based on the information, this sub-module creates the graph of the whole network and finds the optimal paths from source to destination. Depending on the required optimality, the shortest paths can be obtained using Dijkstra's algorithm.

Weight Computation Sub-Module: This sub-module calculates the weight of each link by using OpenFlow statistics. The port statistics of each switch are provided by the topology information sub-module, which are then used to calculate the weight on each link. For example, the link utilization from $v_s^{d_i}$ to $v_t^{d_i}$ can be obtained from the real-time port statistics as the bytes received $N(t_e^k)$ during k-th time interval t_e. This is then integrated with the controller's flow information for traffic types. Following this, the weight of each path is computed as a sum of each link on that path. In Figure 5, there are two paths from S1 to S4, and each of these paths has two links. Total weight on each path can be calculated as $W_{p_i} = \sum_{e=0}^{n} W_e$ where, W_e is the weight of each link on path p_i and computed as $W_e = \frac{\sum_{i=1}^{m}(R_\alpha^i + \gamma^i)}{L_A^e}$. Here, R_α^i is the average flow rate of i-th flow class in t_e on the e-th link, γ^i is the standard deviation for the flow rate of i-th flow class on the e-th link, and L_A^e is the available link capacity of the e-th link.

Finally, these weight values are used to find the optimal path between source and destination pair, where less weight value signifies better optimality. Moreover, the list of associated switches is forwarded to the flow management sub-module for flow installation. If the destination does not belong to this domain, and the domain information sub-module has no information for the required destination then this module will compute and forward optimal paths from source to all of the gateways of this domain which is further used in the inter-domain communication module.

Flow Management: Responsibility of Flow Management Sub-Module is to install flow entries on devices forwarded by Weight Computation Sub-Module. It translates the path information into OpenFlow rules and adds or removes them at each switch. First, the new path is installed, and then the old path is removed. Notably, the controller is a centralized entity, thus it can be assumed that it is capable enough to undertake the load of the system. To avoid path oscillations, a threshold β is used relative to the p_i's capacity and can be configured by the system administrator.

The complete process of Intra-domain load balancing is presented in Algorithm 1 where the shortest paths are computed after generating graph G for intra-domain communication. Weight values of each path are calculated by summation of link weights of each path and the path with minimum weight value is considered as the best path. Finally, the list of switches coming under the best path is returned by the algorithm in order to install the flow rules.

V. Inter-Domain Load Balancing and Vertical Extension

This section presents horizontal load balancing and vertical extension for inter-domain communication. If source and destination belong to different domains (horizontally or vertically), then inter-domain communication is required. In the proposed solution, some of the data plane devices (such as Access Points) are delegated partial controller functionalities in vertical extension, thus making them sub-domain controllers for the perception plane device groups. The communication between the domain controller and these sub-domain controllers (with delegated functions) is conceptually similar to

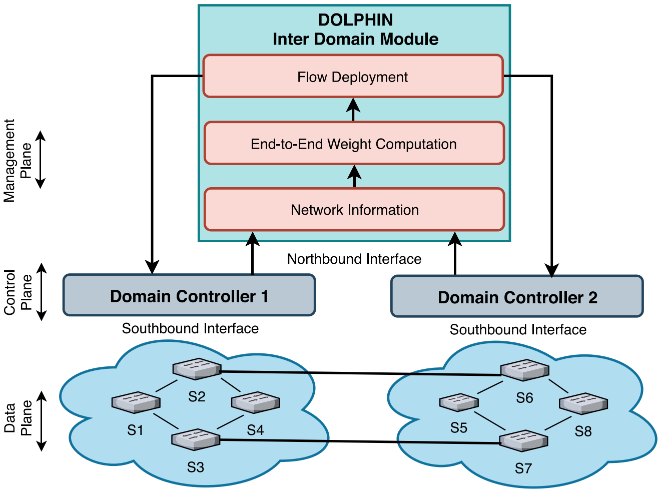


Fig. 6. Inter-domain system and its sub-modules.

Algorithm 2 Inter-Domain Path Selection Process

```
procedure INTER–DOMAIN
    G = ($V^d$,E): network graph
    $V^d$ = set of switches of all domains
    $C^{d_n}$: a set of controllers for n domains
    $P_i$: a set of paths from s to t
    $W_{P_i}$: a set of weight for all paths
    $P_b$: best Path
    $F$: a set of switches to install flows
    $P_i \leftarrow$ G
    for each path $p \in P_i$ do
        $W_{P_i} = \sum_{e=0}^{n} W_e$
    end for
    $P_b = min(P_i, W_{P_i})$
    $F \leftarrow$ list of switches in $P_b$
    for each $C \in C^{d_n}$ do
        for each $F_i \in F$ do
            if $F_i \in V^{d_i}$ then
                $C^{d_i} \leftarrow F_i$
            end if
        end for
    end for
end procedure
```

inter-domain communication. As every domain controller has the information of its domain only hence, generating a detailed multi-domain global graph of the network becomes a challenge. Moreover, each domain controller only optimizes the path within its domain, hence, a gateway switch selected by the source as the optimal exit point, may not be an optimal entry point for the neighboring or destination domain. As all the controllers have equal rights within their domains, hence forming a consensus among them is challenging. Finally, the presence of a non-cooperative domain between source and destination domains is an added challenge.

In horizontal load balancing, if the domain controller receives a packet for a destination that is not under its domain, then the packet is forwarded to the inter-domain communication module. This module uses various sub-modules and collects information from all the controllers by using the REST API of the respective controller to create a global view of the complete network, thus enabling optimal inter-domain communication. To generate the global view for different domains, connectivity of gateway switches of each domain is used. Horizontal inter-domain load balancing is classified into two parts; directly connected domains and indirectly connected domains. Similarly, the vertical extension also involves inter-domain communication, as we discuss later, that the Radio Access Network (RAN) is treated as a domain with a sub-controller at the AP. Similar to the intra-domain dynamic path changing principle, the paths are updated during the lifetime of the flow. Below we discuss each of the modules in detail, and then describe the complete process in Algorithm 2.

A. Directly Connected Domains

In directly-connected domains, gateway switches of source and destination domains are adjacent and connected, as shown in Figure 6. Here two different domain controllers are managing their respective domains. If there is only a single path connecting two domains, then the process is simple. Both domains only have to optimize the paths to the gateway switches in their domains. However, if there is more than one link connecting the domains, then selecting ingress and egress switches and balancing the load on multiple links is difficult. In either case, the proposed Dolphin solution can find the optimal path and balance the load dynamically. It is important to note that, there can be *n* intermediate domains between source and destination. If all of these domains are cooperative, then we still consider the source and destination to be directly connected. In the following subsections, we describe the individual processes of the inter-domain module, as shown in Figure 6.

Network Information Sub-Module: Similar to the domain information process of the intra-domain module, the network information sub-module also has complete information on the data plane devices and links. This information is forwarded to it by the controllers of respective domains as a summary graph. When a flow is required, it first determines all possible paths from the source switch to the destination switch and then appends the network state information with each link. For simplicity, assume the topology as shown in Figure 6. The global graph G = (V^d, E) for these two different domains d_i and d_j can be designed by using following information: $V^{d_i} \in V^d$ is a set of nodes in domain d_i, $GW^{d_i} \in V^{d_i}$ is a set of gateways in domain d_i, $V^{d_j} \in V^d$ is a set of nodes in domain d_j, $GW^{d_j} \in V^{d_j}$ is a set of gateways in domain d_j, $E^{d_i} \in E$ is a set of edges between V^{d_i} & GW^{d_i}, $E^{d_j} \in E$ is a set of edges between V^{d_j} & GW^{d_j}, and $E^{d_{ij}} \in E$ is a set of edges between GW^{d_i} & GW^{d_j}. Once it collects all the required information of switches and gateways from domain controllers, it generates paths from source to destination.

End-To-End Weight Computation Sub-Module: Based on the paths provided by the network information sub-module, the end-to-end weight computation sub-module calculates the weight on the individual links and sums all the weight values to find the total weight of each path. Furthermore, it computes the optimal path from source to destination by comparing these weight values. Path with less weight value is considered as an optimal path, and a list of all associated gateways and switches coming under the best path is forwarded to the deployment module for flow installation. The formulation is the same as that of intra-domain weight calculation, hence we skip it for simplicity.

Deployment Sub-Module: To install the flow entries on switches and gateways, this sub-module communicates with the domain controllers to provide a list of associated switches of each domain controller. Based on this information, each domain controller installs flow entries on each switch to establish the communication between source and destination pair. Similar to the prior explanation, β can be used to avoid path oscillations.

Algorithm 2 presents inter-domain communication where a graph for the whole network is built by taking information from each controller. This graph is further used to find the shortest paths between source and destination pair. Weight values of each path are computed by summation of link weights and path with minimum weight value is considered as the best optimal path. Finally, the list of switches is returned to the respective controllers to install flow rules.

B. Indirectly Connected Domains

Here we cover a scenario where the source and destination domains are connected via an intermediate domain, which does not cooperate in a topology information exchange or flow installation. The data plane in such a scenario is depicted in Figure 7 where the intermediate domain has five switches R1-R5. There can be multiple paths available, however, the Dolphin system has no control over them.

Periodic Probes: The Network Information Sub-Module of the inter-domain communication system periodically sends probe messages across the intermediate non-cooperative domain to estimate the network conditions, such as delay and available throughput. The objective is to find the optimal ingress and/or egress gateway from the neighboring domain. For example, in Figure 7, the network information sub-module sends probe packets from S2 & S3 towards dummy destinations connected to S6 & S8. By observing where the probe packet was received, it determines the potential path and its QoS properties in the non-cooperative domain. Using this information, this model then builds a list of paths with their network conditions, when a flow needs to be created over the said domain. It is important to note, that it is not possible to dictate a specific path within the non-cooperative domain, hence only the selection of ingress/egress switches is of importance to the proposed solution. If there is only one such pair, then he solution does not use probes, while the frequency of probes in the other cases is very less. From experimentation, we have observed that these probes have no performance effect on the links.

Flow Creation: The End-to-End Weight computation modules use the information gathered by probes, in the previous process to estimate the weight of using a specific egress switch (e.g., S2 or S3) when finding the optimal paths, using intra-domain formulas.

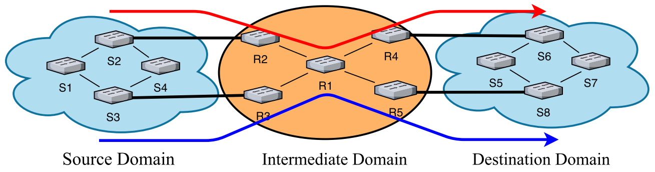


Fig. 7. Data plane of indirectly connected domains.

Flow Installation: The Deployment sub-module installs the flow from the source switch to the egress switch based on the optimal weight value as determined in the previous step (i.e., S2 or S3 in the example). However, it is important to install the rest of the flow on all possible re-entry points in the next cooperative domain. Hence, in Figure 7 the flow entries are made on both S6 and S8, as it is not possible to determine the behavior of the intermediate domain. Even during communication, the path may change, hence, the flow entries must be present for continued forwarding of flow. However, this only occurs when there are multiple gateways connecting the intermediate domain.

C. Vertical Extension

To enable the vertical extension and allow the controllers to have topological information beyond the SDN switches (into the perception plane), the proposed solution creates hybrid devices at the network access points. These devices are considered as secondary controllers, with the ability to discover the topology and configure underlying devices only. Hence, the partial control functions of the controller are delegated to the APs. This scenario becomes very similar to a generic inter-domain communication situation, with the only difference is that the secondary controller (or delegated control device) has limited capabilities. It is important to note, that in this solution, we assume that the underlying perception plane devices can communicate with each other and form an ad-hoc network. This assumption is realistic, as V2V, M2M, and 5G communication allows such multi-hoping in the mobile domain. In our implementation and evaluation, we have achieved this through similar programming of devices and custom wrapper classes for forwarding rules in the routing tables. Figure 8 shows the complete topological structure and the secondary controller's control delegation. Here, we explain the working of inter-domain sub-modules in relation to this delegation.

Network Information Sub-Module: This module receives the topological information from controllers of different domains. In addition to this, it also receives the topological information from secondary controllers at APs. Each AP uses a specialized SBI for sending this information. It is important to note that this information includes, device details, interface types (data rate and technology), and connectivity. In the proposed system, we have enabled all mobile devices to run a piece of simple information reporting app, which updates the secondary controller with the desired information. After receiving this periodic information, the AP generates a sub-graph and sends it to the Network Information Sub-Module, where it

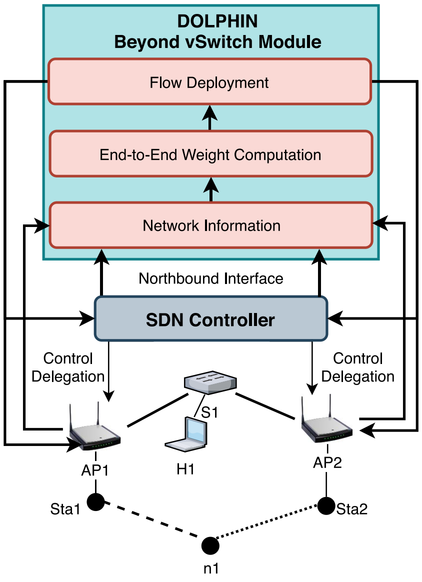


Fig. 8. Vertical extension beyond SDN switch.

is merged with the global topology graph to find optimal paths.

End-To-End Weight Computation Sub-Module: After computing all possible paths between data plane and perception plane elements, the weight value of each path is calculated. Unlike horizontal load balancing, weight computation in vertical extension is different because due to the involvement of different user level devices. Hence, $W_e = \frac{\sum_{f=0}^{n}(R_{\alpha}^{f})}{L_{S}^{e}}$ is used, without any traffic class. This is because only mice flows are allowed to traverse the perception plane links and R_{α}^{f} is the average flow rate of f-th flow using that link at the measurement time. Moreover, L_{S}^{e} is the leftover link capacity from a specific shared bandwidth value allowed by the device. After computing the best path, a list of switches is forwarded to associated controllers and sub-controllers for configuring the associated switches and devices, respectively.

Deployment Sub-Module: Similar to previous sections, this module installs the flows on switches and devices. Figure 8 presents a topology where APs are working as secondary controllers as well. Two mobile stations (i.e., sta1 and sta2) are connected with these APs, and at the same time by using different interfaces these stations are connected with an end device n1. In order to transfer data between h1 and n1, there are two possible paths as S1 → AP1 → Sta1 → n1 and S1 → AP2 → Sta2 → n1. Both the APs and SDN controller forward all the connectivity information of their respective domains to the network information module. Based on this information, the global network view is generated which helps in finding the best optimal path. After weight computation, flow entries are forwarded to respective controllers, which then use the customized APIs to configure the routing and forwarding tables of the devices in the perception plane. It is important to note that such modification to device tables is possible in the majority of open-source systems.

VI. Performance Evaluation and Analysis

The performance of the proposed architecture and solution is evaluated in several scenarios. First, we present a proof of work, with a limited topological setup in Mininet [39] and its fork Mininet-WiFi [40] to evaluate the soundness of the approach. However, as this is only proof of concept and must be translated to real applications, in the second set of experiments, we implement the solution in a data center network scenario with multiple controllers to evaluate the flow completion times. This is a multi-domain environment with a diverse set of data traffic. Finally, we evaluate the extended perception plane architecture in a 5G vehicular network scenario to show the flow completion and path changes in a highly dynamic topology.

A. Proof of Concept

To evaluate the basic working principle of the Dolphin solution we have evaluated it in 4 different scenarios. The network is modeled in Mininet and Mininet-WiFi to mimic the 5G access, edge, and core networks. It is important to note here, that the objective of this work is not to evaluate 5G communication, rather we evaluate the effect of multi-domain SDN in a 5G environment.

In our evaluations, we have used OpenDayLight (ODL) [41] Beryllium-SR4 and FloodLight (FL) [42] controllers, both written in JAVA. FloodLight is a modularized and extensible controller with large community-based support. It uses the OpenFlow protocol to orchestrate flows in the SDN environment. OpenDaylight, on the other hand, provides extensive flexibility in a distributed environment. It is a collection of OSGi bundles that run as Apache Karaf components. Additionally, we deploy the extended Dijkstra's algorithm proposed in mED-SDN [26] to compare our work regarding performance metrics such as RTT and link utilization. Although the performance evaluation for both works has been initialized in the Mininet-based test environment, the performance evaluation of [26] was limited to Abilene as the testbed topology, and ODL as a centralized controller. It is important to note that only ODL provides multi-controller support, hence the evaluation has been limited to it in the inter-domain experiments. Moreover, switch migration solutions are not comparable to the proposed multi-domain/vertical extension solution. Dolphin is implemented as a Python application and uses REST APIs as the northbound interface. The simulation parameters are listed in Table II, and the hosts only generate the traffic and are not considered for weight calculations.

1) Intra-Domain Communication: The initial scenario consists of a single SDN controller which can be realized as shown in Figure 9. There are four switches and two hosts connected at switches S1 and S3, respectively. It can be observed that there are two possible paths between the source and the destination. This scenario determines if the proposed solution can maintain the optimal path and redirect traffic on a less loaded path dynamically. The main evaluation parameters in this scenario are the Round Trip Time (RTT) and the link utilization of different paths when H1 transfers data to H2. Figure 10 presents

TABLE II
SIM. PARAMETERS FOR PROOF-OF-CONCEPT EXPERIMENTS

Parameter	**Value**
Platform	Mininet (with OVS), Mininet-WiFi
Topology	Figs. 9, 12a, 13a
Link bandwidth	10 Gbps
Traffic types	Observed flow (UDP) Domain traffic (UDP)
Host pairs	1 per topology
Flow rate	50 per sec
Per flow data rate	50 Mbps
Link latency	2 μs
Simulation time	20 sec. (without setup)

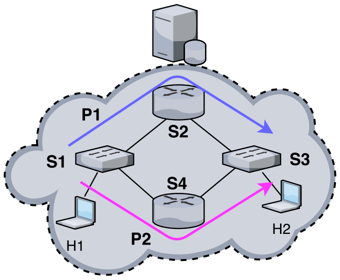
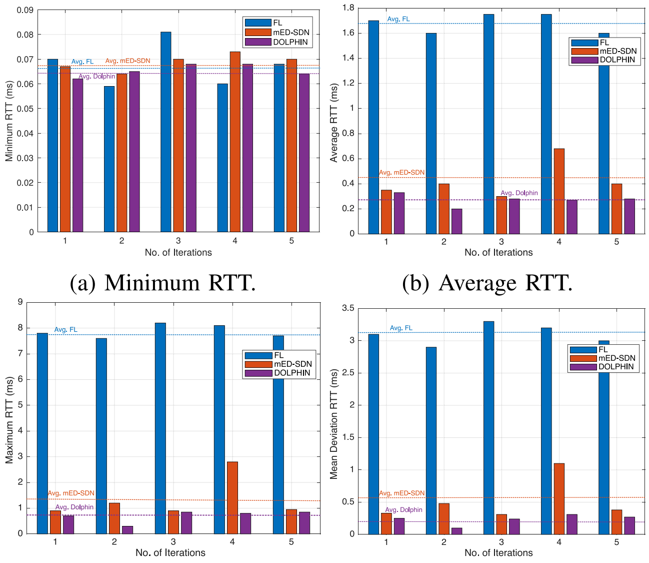


Fig. 9. Intra-domain topology.



(a) Minimum RTT.

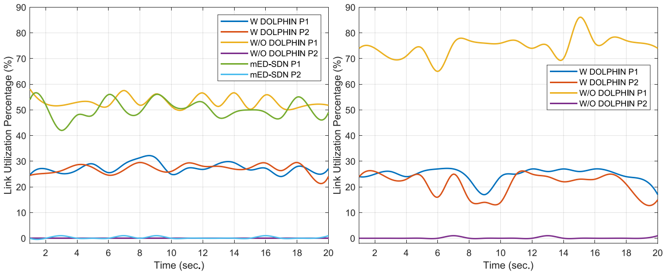
(b) Average RTT.

(c) Maximum RTT.

(d) Mean Deviation of RTT.

Fig. 10. RTT for intra-domain using FloodLight.

the minimum, average, maximum, and mean deviation RTT against floodlight controller and mED-SDN, while Figure 11 shows the link utilization against different controllers. The experiments have been repeated 10 times, and average results are presented.



(a) Against ODL and mED-SDN.

(b) Against FloodLight.

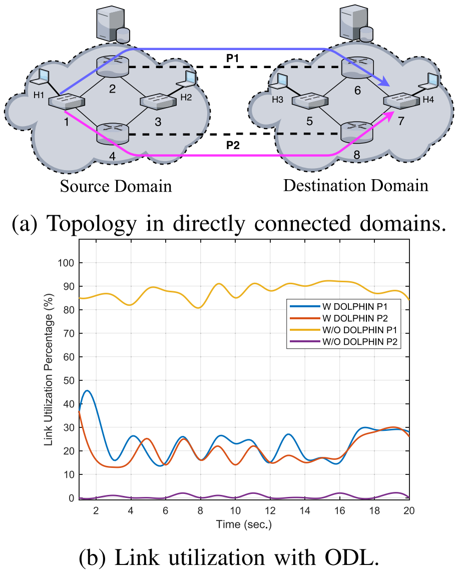
Fig. 11. Link utilization for intra-domain module.

It can be observed from Figure 10, that there is a minor difference in minimum RTT between FloodLight controller, mED-SDN, and Dolphin. Average RTT is more than 1.5 ms with FloodLight controller whereas, mED-SDN performs better as compared to Floodlight and takes 0.3 ms to 0.6 ms. Dolphin outperforms both of the existing solutions and gives 0.25 ms average RTT. Similarly, maximum RTT is less than 1 ms, and the mean deviation is less than 0.3 ms with the proposed solution.

To compute link utilization on all possible paths from source to destination, we created two applications. One application generates the observed data flow, while the other generates random traffic across the network to model load on links. It can be observed from Figure 11 that without Dolphin the usage of the initial path is maximum, whereas the second path is neglected by using the OpenDaylight controller. For example, path *P1* is overloaded with more than 50% utilization, whereas usage of path *P2* is zero for the observed flow. It is important to note that the load generating application directs traffic randomly on the different links, hence it is not possible that the other path is overloaded with it. Similar to ODL, mED-SDN results show that its behavior is almost similar. Contrary to these, the proposed algorithm dynamically switches the path of the observed flow and utilizes both the links evenly. It can be seen that the link utilization is less than 30% for the observed flow on both links. Similarly, when the Floodlight controller is used, link utilization of one path is more than 70% and the other link is not being used at all. However, both of these links are being used to transfer data after running Dolphin and overall link utilization has been reduced for the observed flow. Notably, due to multiple cost functions of weight value, there is no link flapping while Dolphin is used.

2) Communication Among Directly Connected Domains: The scenario to evaluate inter-domain (but directly connected) load balancing is shown in Figure 12(a). It can be seen that there are multiple paths from the source domain to the destination domain, and within each domain, there are multiple paths from the egress/ingress switch to the destination switch. It is important to note that we have evaluated this using ODL only, as the Floodlight controller is entirely centralized, and does not work with multiple domains. Figure 12(b) presents the performance of Dolphin in terms of link utilization. It is evident that without the use of any load balancing system, the initially selected path is utilized throughout the duration of flow, while the other path has free bandwidth available. However, with the use of the proposed algorithms, both of the paths are utilized at the rate of 30% to 40% by the observed flow. Reduction of link utilization and load on both links proves that the load is distributed evenly and resources are being utilized efficiently.

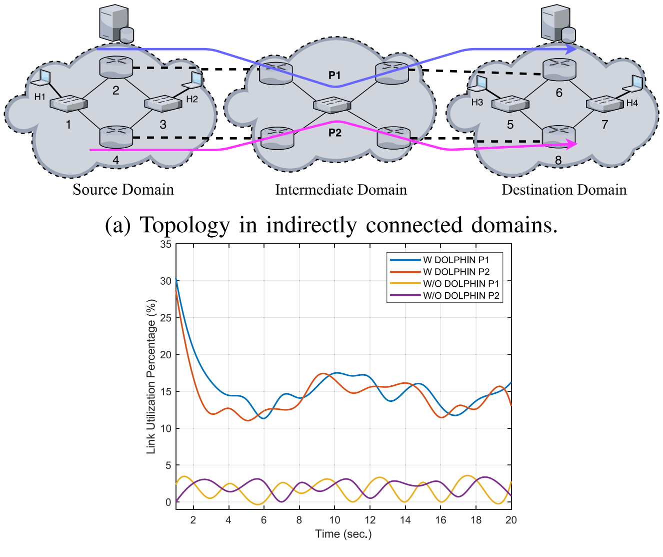
3) Communication Among Indirectly Connected Domains: The third scenario uses almost the same topology as in the



(a) Topology in directly connected domains.

(b) Link utilization with ODL.

Fig. 12. Inter-domain topology and link utilization.



(a) Topology in indirectly connected domains.

(b) Link utilization with ODL.

Fig. 13. Indirectly connected domains and link utilization.

previous experiment, but with an intermediate domain as presented in Figure 13(a). This intermediate domain does not provide any internal topological information and does not allow the proposed scheme to install paths on its switches. However, it uses a heuristic approach for the flows to transit through its network. For simplicity, we assume that the possible paths for observed flow are P1 and P2. Furthermore, the intermediate domain has some internal traffic so that the flows are disrupted. Figure 13(b) presents the link utilization for indirectly connected domain scenario. Due to changes in the uncontrolled intermediate domain, both links are used without applying Dolphin but this utilization is less than 5%. It is important to note that the link utilization shown here does not include the path-segment from within the non-cooperative intermediate domain. Hence, the links from H1 to egress of source domain (switch 4) and then switch 8 to H4 are the segments which are measured, which remains low due to the generic method. However, link utilization is improved for the observed flows to 10% after running Dolphin.

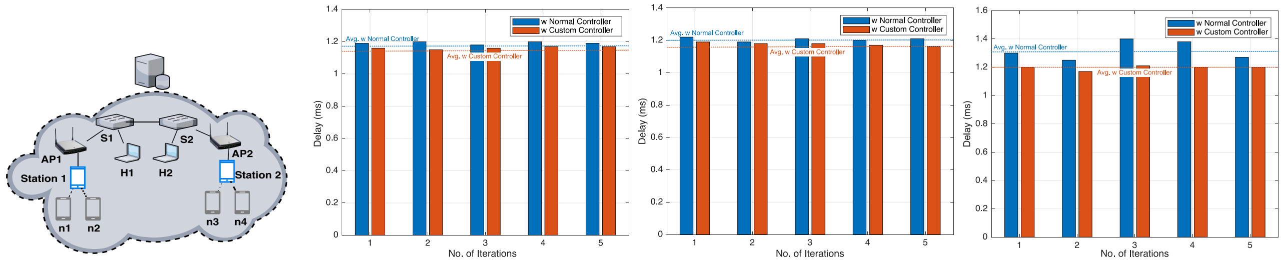
4) Communication Among Data Plane and Perception Plane Elements: Particularly for this scenario, we have used Mininet-WiFi to enable wireless communication. Mininet-WiFi is a fork to Mininet and extends its functionality by adding different classes to support stations and access points. In order to delegate control, we have modified Mininet-WiFi where an AP can act as a sub-controller. Moreover, the mobile stations are also made capable of interacting with controllers through a specialized SBI. Figure 14(a) presents the topology used for this scenario. Due to the delegation of controller functionalities to access points, it becomes an inter-domain communication. To evaluate vertical extension, we compute delay and evaluate by comparing the performance of a custom controller (i.e., secondary controller) and performance of directly connected domains in Mininet-WiFi.

Moreover, Figure 14 shows the delay with the custom and normal controller. The custom controller presents the delay when control is delegated to data plane elements to handle the perception plane. Whereas, normal controller delay shows a traditional SDN controller to handle data plane elements. Figure 14(b) presents delay when both of these controllers handle only 2 nodes. In Figures 14(c) and 14(d) number of nodes are increased by one. In all three cases, the delay with custom controller and normal controller is almost similar, which shows that after control delegation no additional delay is introduced. The controller load is presented in Figure 15, where load is computed by using number of requests sent by each switch. It can be noticed that load is reduced slightly when control is delegated to data plane elements by using Dolphin. As communication among different devices at perception plane is handled by data plane elements which also work as controller thus, it reduces the load over the main controller.

B. Multi-Domain Datacenter Communication

We have implemented the proposed Dolphin algorithm in a simulated environment using Mininet and then raw data output has been processed in MATLAB to evaluate scalability, response to realistic traffic, and overall performance.

The network topology is designed to represent a 4-post leaf-spine architecture, with a core router at the top, as shown in Figure 16(a). The whole network consists of 4 such clusters as shown in Figure 16(b). Each cluster mimics a domain, where the controller is running at the core router. The workloads used in experimentation are similar to the one in literature [43]–[45], which are derived from actual data centers. We use three different types of services: Web search, Data mining, and Hadoop. The traffic distribution of these services is shown in Figure 17. Each server in topology initiates connections for destinations in rack, in cluster, and cross-cluster with even distribution. The service type is also uniformly distributed for connections. All flows have associated completion deadlines. As part of the experiment, we have enabled the Dolphin algorithm to prioritize the flow installation of those flows which have higher priority (i.e., shorter deadline).



(a) Vertical extension topology. (b) 2 Nodes (c) 3 Nodes (d) 4 Nodes

Fig. 14. Topology and average delay with different number of nodes in the vertical extension. Delay is measured end-to-end.

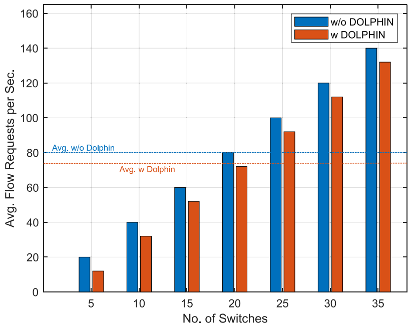
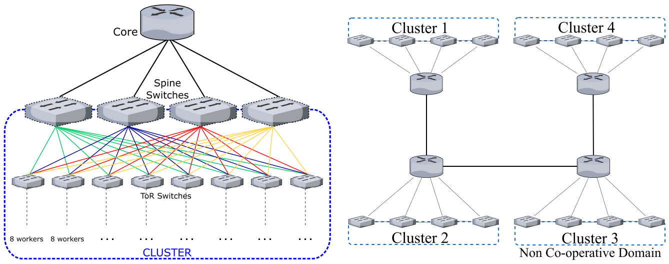


Fig. 15. Controller load with vertical extension.



(a) Cluster Topology. (b) Core connecting 4 clusters.

Fig. 16. Topological connectivity of data center.

TABLE III
SIM. PARAMETERS FOR DATACENTER EXPERIMENTS

Parameter	Value
Topology	4-Post Leaf-Spine (Fig. 16)
Core routers	4
Clusters/Domains	4
Spin switches	4 per cluster
Leaf ToR switches	8 per cluster
Worker servers	8 per rack
Link capacity	10 Gbps
Link latency	2 μs
Non-coop. domain	Cluster 3 (Fig. 16b)
Max host pairs	7680
Traffic types	3 (Fig. 17)
Flow rate	30 per worker
Simulation time	30 sec. (without setup)

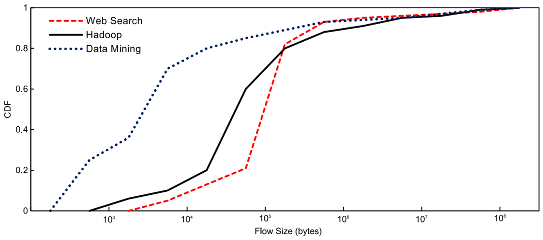
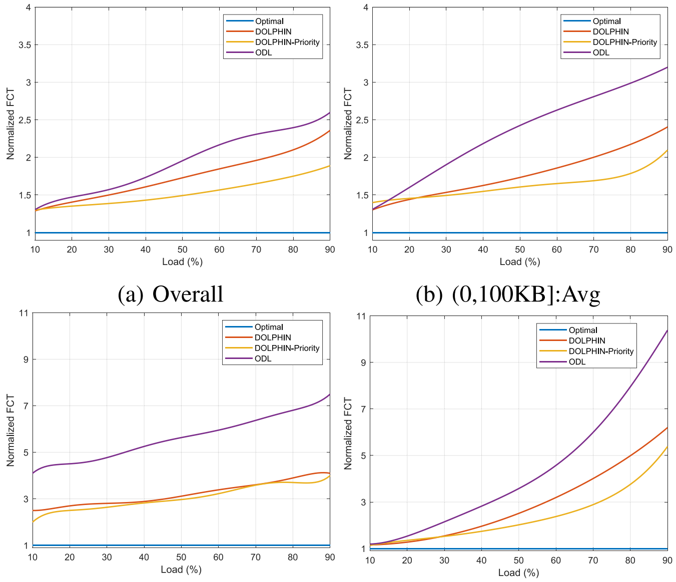


Fig. 17. Traffic distribution of three services for simulation.

We have evaluated the performance of Dolphin in terms of the Flow Completion Times (FCT) as the main metric for comparison. We show the Normalized FCT for small flows, large flows, and for 99th percentile. The simulation parameters are listed in Table III.

The normalized flow completion times for all flows aggregated is shown in Figure 18. The overall FCT in Figure 18(a) shows a clear improvement in reaching deadlines within the given amount of time. At lighter loads on the network, the FCTs are comparable for all the controllers, but as the load on the network increases the congestion also increases. Figures 18(b) & 18(c) show mice flow completion as average, and at 99th percentile (note the change in scale). The performance increase is quite evident especially with a high number of connections in the topology against generic ODL. The large throughput sensitive flows also show improved completion time of orders of magnitude more in Figure 18(d). The load balancing of new flows (for congested destinations) helps in reducing the congestion quickly at intermediate points, while the existing flows are switched to paths with less load. The knowledge of the network state helps in picking the appropriate path and dynamically change it using Dolphin. When Dolphin is made priority aware, it dynamically adjusts the flows for shorter deadline flows before the other regular flows. Hence, they experience better FCT as compared to the rest.

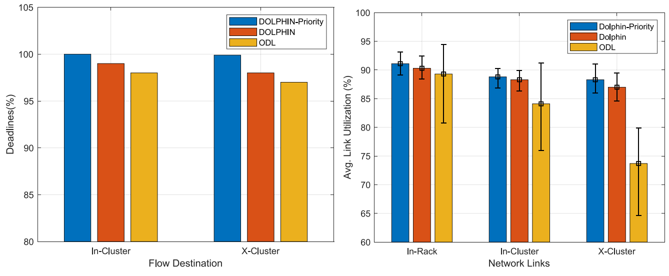
Figure 19 shows the completed flows within their deadlines and the average link utilization of the network within the rack, within the cluster (ToR to spine), and across the cluster. Figure 19(a) shows the percentage of completed flows within deadlines at different locations of the network. The performance of flows within the rack is almost similar in each case, as the possible paths are short and not many congested points are available. It becomes more prominent in cross-cluster communication and within the cluster, where there may be more congestion points. Dolphin has shown to complete 99.5% of the flows within the deadline. In Figure 19(b) we show the average utilization of the links, which gives an estimation of the load balancing through the minimum and maximum deviations. It needs to be clarified here, that the



(a) Overall (b) (0,100KB]:Avg

(c) (0,100KB]:99^{th} Percentile (d) (10MB,∞):Avg

Fig. 18. Normalized FCTs for all services.



(a) Completed flows in deadline. (b) Average link utilization.

Fig. 19. Flow completion and link utilization percentage.

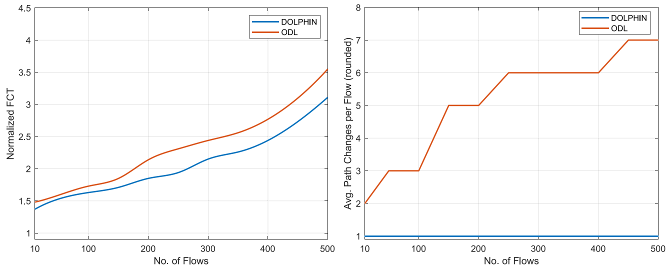
error lines are depicting minimum and maximum deviation in link utilization from the average. We can observe that the deviation is less when Dolphin is used, and more when generic ODL is working. Moreover, the overall link utilization for ODL is also less as compared to the Dolphin solution.

C. 5G Vehicular Network

In this section, we evaluate the performance of the proposed scheme for the vertical extension in 5G vehicular network environment. 5G networks extended for vehicular communication are also similar to inter-domain communication, where the Road Side Units (RSUs) connected through next-generation Node-B (gNB) to edge clouds [46]. These clouds are highly programmable and can have their own controllers. However, the controller can only view the gNB, while the perception plane (RSUs and vehicles with V2X communication) is not visible (or programmable). Hence, in this experiment we model the vertical extension of Dolphin, enable V2I and V2V communication [47]. It is important to note, that the objective is not to evaluate the 5G communication, but use a 5G environment to model the connectivity of V2V and V2I systems. Hence, we have used a hybrid modeling environment. The topology and connectivity has first been built using NS-3 [48], and then time-stamped connectivity and data rate statistics of vehicles have been ported to Mininet-WiFi, where the hosts are modeled based on these input parameters. In Figure 20 we show the FCT and average path changes of topology, which has 3 edge clouds (i.e., domains) and parameters as shown in Table IV. For simplicity, all domains are cooperative.

TABLE IV
SIM. PARAMETERS FOR VEHICULAR NETWORK EXPERIMENTS

Parameter	Value
Platform	Mininet-WiFi, NS-3
Topology	Trace data [49]
Edge Clouds/Domains	3
RSUs	10 per cloud
Ingress/Egress RSUs	3 per cloud
No. of vehicles	53
RSU Range	100 m
No. of Flows	10-500
Simulation time	30 sec. (without setup)



(a) Average FCT. (b) Average path changes.

Fig. 20. Perf. of vertical extension using vehicular network.

Figure 20(a) shows the normalized FCT of all flows. It can be observed that by using the vertical extension of the proposed scheme, we can improve the completion times, as the flows within the V2X communication can be programmed and dynamically optimized. In Figure 20(b) we present the number of average path changes with the increasing number of flows. It is important to note that once the path is selected in traditional controllers it is not changed until the flow completes. Hence, ODL has only 1 path per flow. However for the proposed solution, as there are a number of flows, the possible path changes become more frequent. It is obvious that more path changes will mean more control information packets (flow installation packets) in the network. However, given the clear improvement in flow completion times, we believe that the control overhead is acceptable.

VII. Conclusion

In real-world networks, redundant paths are available between any pair of communicating devices, especially when they exist in different network domains. In most of the scenarios, these network resources in terms of link utilization are wasted because the flows are not dynamically managed. In this work, we have proposed a dynamic solution for SDN controllers which load balances the flows among multiple links dynamically across different domains. Furthermore, it also provides better resource utilization in case of an intermediate domain which is not co-operative. The solution gives dynamic control over network elements beyond the traditional virtual switches, in the perception plane, such as IoT devices, sensors, and mobile stations. Moreover, the algorithms developed in this work are also applied to vehicular communication, which is yet another example of an extended perception plane with unique requirements and different domains. The experimental results first establish that the solution is viable through a simple testbed, and then rigorous testing on the data center environment and 5G vehicular network further prove its capabilities.

Several future research directions can be taken for the work proposed in this article. The extension of perception plane devices may include different physical communications standards and their tight integration into the topological graph and Openflow data can be an interesting optimization. Similarly, intelligent flow rule modification at the access points, and understanding of Openflow can prove to be beneficial. Currently, the interfacing APIs between the master controller and delegated-control devices only work for the designed system. These APIs can be further extended and standardized to be more capable. Implementation of similar work in the UAV domain is also possible and may prove to be extremely valuable, and further experimentation can also be beneficial to evaluate performance.

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

[3] S. Saraswat, V. Agarwal, H. P. Gupta, R. Mishra, A. Gupta, and T. Dutta, "Challenges and solutions in software defined networking: A survey," *J. Netw. Comput. Appl.*, vol. 141, pp. 23–58, Sep. 2019.

[4] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A comprehensive survey of interface protocols for software defined networks," *J. Netw. Comput. Appl.*, vol. 156, Apr. 2020, Art. no. 102563.

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

[6] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Sep. 2013.

[7] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM*, 2013, pp. 15–26.

[8] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017.

[9] K. Kuroki, M. Fukushima, M. Hayashi, and N. Matsumoto, "Redundancy method for highly available OpenFlow controller," *Int. J. Adv. Internet Technol.*, vol. 7, no. 1, pp. 114–123, 2014.

[10] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX Conf. Oper. Syst. Design Implement.*, 2010, pp. 351–364.

[11] 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.

[12] L. Zhu *et al.*, "SDN controllers: A comprehensive analysis and performance evaluation study," *ACM Comput. Surveys*, vol. 53, no. 6, pp. 1–40, Dec. 2020.

[13] S. Bera, S. Misra, and A. V. Vasilakos, "Software-defined networking for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1994–2008, Dec. 2017.

[14] I. Alam *et al.*, "A survey of network virtualization techniques for Internet of Things using SDN and NFV," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–40, 2020.

[15] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[16] (2017). *POX*. Accessed: Oct. 25, 2020. [Online]. Available: https://github.com/noxrepo/pox

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

[18] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed SDN controller," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, Oct. 2014, pp. 17–27.

[19] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. Workshop Hot Topics Softw. Defined Netw.*, 2012, pp. 19–24.

[20] H. Long, Y. Shen, M. Guo, and F. Tang, "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks," in *Proc. IEEE 27th Int. Conf. Adv. Inf. Netw. Appl.*, Mar. 2013, pp. 290–297.

[21] A. M. Al-Sadi, A. Al-Sherbaz, J. Xue, and S. Turner, "Routing algorithm optimization for software defined network WAN," in *Proc. Al-Sadeq Int. Conf. Multidiscipl. IT Commun. Sci. Appl.*, May 2016, pp. 1–6.

[22] M. F. Ramdhani, S. N. Hertiana, and B. Dirgantara, "Multipath routing with load balancing and admission control in software-defined networking (SDN)," in *Proc. 4th Int. Conf. Inf. Commun. Technol.*, May 2016, pp. 1–6.

[23] A. Khaliq, S. H. Adil, and J. Jamshid, "Enhancing throughput and load balancing in software-defined networks," in *Proc. Int. Conf. Comput. Math. Eng. Technol.*, Mar. 2018, pp. 1–6.

[24] D. Adami, S. Giordano, M. Pagano, and G. Portaluri, "A novel SDN controller for traffic recovery and load balancing in data centers," in *Proc. IEEE Int. Workshop Comput. Aided Model. Design Commun. Links Netw.*, Oct. 2016, pp. 77–82.

[25] Y.-L. Lan, K. Wang, and Y.-H. Hsu, "Dynamic load-balanced path optimization in SDN-based data center networks," in *Proc. 10th Int. Symp. Commun. Syst. Netw. Digit. Signal Process.*, Jul. 2016, pp. 1–6.

[26] A. Abdulaziz, E. A. Adedokun, and S. Man-Yahya, "Improved extended Dijkstra's algorithm for software defined networks," *Int. J. Appl. Inf. Syst.*, vol. 12, no. 8, pp. 22–26, 2017.

[27] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2013, pp. 2211–2219.

[28] J. Wang, G. Shou, Y. Hu, and Z. Guo, "A multi-domain SDN scalability architecture implementation based on the coordinate controller," in *Proc. Int. Conf. Cyber Enabled Distrib. Comput. Knowl. Disc.*, Oct. 2016, pp. 494–499.

[29] H. Wang, H. Xu, L. Huang, J. Wang, and X. Yang, "Load-balancing routing in software defined networks with multiple controllers," *Comput. Netw.*, vol. 141, pp. 82–91, Aug. 2018.

[30] W. Lan, F. Li, X. Liu, and Y. Qiu, "A dynamic load balancing mechanism for distributed controllers in software-defined networking," in *Proc. Int. Conf. Measuring Technol. Mechatronics Autom.*, Feb. 2018, pp. 259–262.

[31] Y. Zhou *et al.*, "A load balancing strategy of SDN controller based on distributed decision," in *Proc. IEEE Int. Conf. Trust Security Privacy Comput. Commun.*, Sep. 2014, pp. 851–856.

[32] K. Hikichi, T. Soumiya, and A. Yamada, "Dynamic application load balancing in distributed SDN controller," in *Proc. 18th Asia–Pac. Netw. Oper. Manag. Symp.*, Oct. 2016, pp. 1–6.

[33] P. Sun, Z. Guo, G. Wang, J. Lan, and Y. Hu, "Marvel: Enabling controller load balancing in software-defined networks with multi-agent reinforcement learning," *Comput. Netw.*, vol. 177, Aug. 2020, Art. no. 107230.

[34] M. Bagaa, D. L. C. Dutra, T. Taleb, and K. Samdanis, "On SDN-driven network optimization and QoS aware routing using multiple paths," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4700–4714, Jul. 2020.

[35] Z. Duliński, G. Rzym, and P. Chołda, "MPLS-based reduction of flow table entries in SDN switches supporting multipath transmission," *Comput. Commun.*, vol. 151, pp. 365–385, Feb. 2020.

[36] S. Bera, S. Misra, and A. Jamalipour, "FlowStat: Adaptive flow-rule placement for per-flow statistics in SDN," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 530–539, Mar. 2019.

[37] P. R. Torres, Jr., A. García-Martínez, M. Bagnulo, and E. P. Ribeiro, "Bartolomeu: An SDN rebalancing system across multiple interdomain paths," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107117.

[38] M. Hamdan *et al.*, "A comprehensive survey of load balancing techniques in software-defined network," *J. Netw. Comput. Appl.*, vol. 174, Oct. 2020, Art. no. 102856.

[39] (2015). *Mininet: An Instant Virtual Network on Your Laptop (or Other PC)*. Accessed: Oct. 25, 2020. [Online]. Available: http://www.mininet.org/

[40] (2017). *MiniNet-WiFi: Emulator for Software-Defined Wireless Networks*. Accessed: Oct. 25, 2020. [Online]. Available: https://github.com/intrig-unicamp/mininet-wifi

[41] *Opendaylight: A Linux Foundation Collaborative Project*. Accessed: Oct. 25, 2020. [Online]. Available: https://www.opendaylight.org

[42] (2014). *Project FloodLight*. Accessed: Oct. 25, 2020. [Online]. Available: https://github.com/floodlight/floodlight

[43] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010.

[44] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2013, pp. 2157–2165.

[45] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

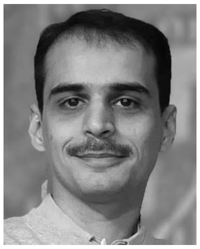
[46] "View on 5G architecture," Working Paper, 5G PPP Archit., Valencia, Spain, Jun. 2019. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2019/07/5G-PPP-5G-Architecture-White-Paper_v3.0_PublicConsultation.pdf

[47] W. B. Jaballah, M. Conti, and C. Lal, "Security and design requirements for software-defined VANETs," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107099.

[48] (2020). *Network Simulator (NS-3)*. Accessed: Oct. 25, 2020. [Online]. Available: https://www.nsnam.org/

[49] Y. Zheng. (2011). *T-Drive Trajectory Data Sample*. Accessed: Oct. 25, 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

Zohaib Latif received the B.S. degree in electrical engineering and the M.S. degree in electrical and electronics engineering from the University of Glasgow, U.K., in 2006 and 2008, respectively, and the Ph.D. degree from the School of Computer Science, Beijing Institute of Technology, Beijing, China. Since 2011, he has been working as a Senior Lecturer with Beijing Institute of Technology. He is currently working as an Assistant Professor with the Department of Computing, Riphah International University, Faisalabad Campus. His major interests are in software-defined networks (SDN), distributed controllers in SDN, and Internet of Things.



Kashif Sharif (Senior Member, IEEE) received the M.S. degree in information technology from the National University of Sciences and Technology, Pakistan, in 2004, and the Ph.D. degree in computing and informatics from the University of North Carolina at Charlotte, Charlotte, NC, USA, in 2012. He is currently an Associate Professor for research with Beijing Institute of Technology, Beijing, China. His research interests include data centric networks, blockchain and distributed ledger technologies, wireless and sensor networks, software-defined networks, and 5G vehicular and UAV networks. He is serves as an Associate Editor for IEEE Access. He also serves on several TPCs of IEEE and ACM conferences.

Fan Li (Member, IEEE) received the B.Eng. and first M.Eng. degrees in communications and information system from Huazhong University of Science and Technology, Wuhan, China, in 1998 and 2001, respectively, the second M.Eng. degree in electrical engineering from the University of Delaware, Newark, DE, USA, in 2004, and the Ph.D. degree in computer science from the University of North Carolina at Charlotte, Charlotte, NC, USA, in 2008. She is currently a Professor with the School of Computer Science, Beijing Institute of Technology, Beijing, China. She has more than 100 publications in reputed journals and conferences. Her current research focuses on wireless networks, ad hoc and sensor networks, and mobile computing. Her papers have won Best Paper Awards from IEEE MASS in 2013, IEEE IPCCC in 2013, ACM MobiHoc in 2014, and Tsinghua Science and Technology in 2015. She is a Member of the ACM.

Md Monjurul Karim received the B.Eng and M.Eng degrees in computer science from Northwestern Polytechnical University, Xian, China, and is currently pursuing the Ph.D. degree in computer science and technology with Beijing Institute of Technology, Beijing, China. His research interests include software-defined networking, information-centric networking, named data networks, and next-generation networking.

Sujit Biswas (Member, IEEE) received the M.Eng. degree in computer engineering from Northwestern Polytechnical University, China, in 2015, and the Ph.D degree in computer science and technology from Beijing Institute of Technology, China. He is also an Assistant Professor with the Computer Science and Engineering Department, Faridpur Engineering College, University of Dhaka, Bangladesh. His basic research interest is in Internet of Things, blockchain, mobile computing security and privacy, big data, machine learning, and data driven decision making.

Madiha Shahzad (Member. IEEE) received the M.S. degree in information technology from the National University of Science and Technology, Pakistan, in 2004, and the Ph.D. degree in computer science from the University of the West of England, U.K., in 2012. She is currently working as an Associate Lecturer and as a Postdoctoral Scholar with the School of Sciences, UCLan Cyprus. She has actively participated in several EU FP6/FP7 Research Projects and authored numerous peer-reviewed journal publications, conference papers, and book chapters. Her current research interests include evolving telecommunication systems especially 5G, Internet of Things, QoS aspects, and the considerations of responsible research and innovation in these domains.

Saraju P. Mohanty (Senior Member, IEEE) received the bachelor's degree (Hons.) in electrical engineering from Orissa University of Agriculture and Technology, Bhubaneswar, in 1995, the master's degree in systems science and automation from the Indian Institute of Science, Bengaluru, in 1999, and the Ph.D. degree in computer science and engineering from the University of South Florida, Tampa, in 2003. He is currently a Professor with the University of North Texas. His research is in "Smart Electronic Systems" which has been funded by National Science Foundations, Semiconductor Research Corporation, U.S. Air Force, IUSSTF, and Mission Innovation. He has authored 350 research articles, four books, and invented four granted and one pending patents. His Google Scholar H-index is 38 and i10-index is 147 with 6500 citations. He is regarded as a Visionary Researcher on smart cities technology in which his research deals with security and energy aware, and AI/ML-integrated smart components. He introduced the secure digital camera in 2004, with built-in security features designed using hardware-assisted security or security by design principle. He is widely credited as the designer for the first digital watermarking chip in 2004, and first the low-power digital watermarking chip in 2006. He is a recipient of 12 best paper awards, Fulbright Specialist Award in 2020, IEEE Consumer Technology Society Outstanding Service Award in 2020, the IEEE-CS-TCVLSI Distinguished Leadership Award in 2018, and the PROSE Award for Best Textbook in Physical Sciences and Mathematics Category in 2016. He has delivered ten keynotes and served on nine panels at various international conferences. He has been serving on the editorial board of several peer-reviewed international journals, including IEEE Transactions on Consumer Electronics, and IEEE Transactions on Big Data. He is the Editor-in-Chief of the *IEEE Consumer Electronics Magazine* (MCE). He has been serving on the Board of Governors of the IEEE Consumer Technology Society, and has served as the Chair of Technical Committee on Very Large Scale Integration, IEEE Computer Society from 2014 to 2018. He is the Founding Steering Committee Chair for the IEEE International Symposium on Smart Electronic Systems, the Steering Committee Vice-Chair of the IEEE-CS Symposium on VLSI, and the Steering Committee Vice-Chair of the OITS International Conference on Information Technology. He has mentored two Postdoctoral researchers, and supervised 12 Ph.D. dissertations, 26 M.S. theses, and 10 undergraduate projects.