IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 2, 2020, accepted June 15, 2020, date of publication June 24, 2020, date of current version July 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004609

cRetor: An SDN-Based Routing Scheme for Data Centers With Regular Topologies

ZEQUN JIA[1,2], YANTAO SUN[1,2], (Member, IEEE), QIANG LIU[1,2], SONG DAI[2], AND CHENGXIN LIU[2]

[1]Beijing Key Laboratory of Transportation Data Analysis and Mining, Beijing Jiaotong University, Beijing 100044, China
[2]School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China

Corresponding author: Yantao Sun (ytsun@bjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61572220.

ABSTRACT The blooming of cloud computing leads to the rapid expansion of data center networks (DCN). Conventional link state routing algorithms like OSPF are widely adopted in data center networks, however, those routing algorithms bring great control overheads and long convergence time. Recently, topology-aware routing methods are considered to be efficient especially in data center networks with regular topologies. Lots of topology-aware routing methods (e.g., Fat-Tree and BCube) have been proposed for specific data center network topologies. This paper first proposes a formalized method to describe regular topologies and a regular Topology Description Language (TPDL) based on this method. TPDL is well designed to accurately define regular network topologies in a clear way leveraging their regularities. Based on the Software-Defined Networking (SDN) technology, this paper also proposes a novel topology-aware routing scheme: cRetor (controller-side REgular TOpology Routing scheme). Different from other topology-aware routing methods, cRetor is a TPDL-based general routing method, which means it is expected to work on different kinds of regular topologies. In this scheme, TDPL files are used as a priori knowledge to build an initial topology in the SDN controllers, which eliminates the process of topology discovery via Link Layer Discovery Protocol (LLDP) and hence relieves the bandwidth and processing burdens on controllers. Besides, we also apply the A-star algorithm to SDN controllers to speed up the routing selection, where TPDL's distance formulas act as the heuristic function. The experimental results show that cRetor outperforms LLDP-based SDN, OSPF and DCell in routing calculation performance, convergence speed, routing overheads and fault tolerance.

INDEX TERMS Data center networks, regular network topologies, topology description language, software-defined networks, topology-aware routing algorithms.

I. INTRODUCTION

With the wide application of cloud computing and big data technologies, the scale of data centers has also increased rapidly, which leads to a higher demand for both communication and management capabilities of data center networks. Numerous approaches have been proposed by researchers to enhance the performance of DCNs. Among these methods, topology-aware routing methods are well-known for their efficiency compared with conventional link state routing methods like OSPF. A topology-aware routing method takes into account the physical layout of the network for calculating routing paths and forwarding packets. In recent years, topology-aware routing methods have obtained a surge of interest from researchers, and lots of related study has been conducted, from the switch-centric approach (such as Fat-Tree [1], HHS [2], VL2 [3], Aspen tree [4] and S2 [5]), to the server-centric approach (like BCube [6] and DCell [7]).

Most of these topology-aware routing schemes leverage the regularity hidden in the physical topology structures, that is, these topologies are usually recursively or iteratively defined. In [1], M.Al-Fares *et al.* have proposed a scalable data center network architecture and a corresponding routing technique for Fat-Tree topology. To take advantage of the structure of the Fat-Tree topology, a specific addressing method and the two-level routing tables are proposed. A clear benefit from the topology-aware methods is that they deliver scalable traffic at much lower costs. Similarly, as a representative of server-centric network architecture, routing

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

algorithms in BCube [6] also leverage BCube's topological property to achieve higher performance under lower cost.

In addition to the dedicated routing algorithms proposed along with the new network topologies, many studies on routing algorithms for these topologies are also conducted. For instance, as one of the four most common datacenter network topologies [8], Fat-tree has attracted many researchers' attention since [1]. SADP [9], SAOP [10], DRB [11] and GRR&IGRR [12] are all efficient packet-based routing algorithms designed for Fat-Tree. Besides, In [13], Zhao *et al.* proposed a port-based source-routing addressing (PSRA) scheme for Fat-tree topology. Based on PSRA, they designed a simple routing algorithm, which leverages the characteristics of PSRA and the regularity of Fat-tree topologies. These routing algorithms focus on a certain network topology, trying their best to fully exploit the characters of this topology structure.

Despite their superior performance and efficiency, all the previously mentioned methods suffer from a serious limitation. Almost all of topology-aware routing and forwarding algorithms are designed for a specific network topology, which causes difficulty in renovating a routing device to support multiple existing topology-aware routing algorithms. To the best of our knowledge, there has been little discussion about a generic topology-aware routing scheme that can be adopted in multiple data center network structures. Therefore, as mentioned in [14], there is a need to design an efficient topology-aware routing protocol for generic DCN topologies.

However, a major problem with generic topology-aware routing methods is they are highly dependent on the topology structures and lack of flexibility in handling temporary link changes and failures. Fortunately, the emergence and development of SDN technology bring new opportunities for enhancing manageability and flexibility in data center networks. It separates the traditionally bundled control and data planes, which brings centralized network control, programmability and reconfigurability in data center networks. With the help of SDN, it becomes easier to introduce new networking abstractions, simplify network management and facilitate network evolution [15].

In the field of data center networking, Google employed SDN for data transferring and syncing among data centers at first. In [16] and [17], they introduced B4 to their data center interconnection for traffic optimization and resource allocation. In the global layer of B4, the traffic engineering central server allocates bandwidth and traffic based on the priorities of flows, controlling the OpenFlow switches through the OpenFlow controller in the middle layer. With centralized traffic engineering, B4 improved the link utilization ratio from 30∼40% to more than 90%, significantly reducing the cost of devices.

While inside data centers, traffic engineering and failure recovery methods also benefit from the introducing of SDN. The global view on the SDN controller assists in traffic optimization and failure detection, and the separation of control and data planes make fine-grained flow scheduling possible. Hedera [18] performs dynamic flow scheduling in a data center network with the SDN technique. Compared with static load-balancing methods, Hedera delivers up to 113% better bisection bandwidth. In the same vein, other work such as Afek and colleagues' work [19], DevoFlow [20], MiceTrap [21], RepFlow [22], OpenQos [23] and DIFFERENCE [24] also tried to optimize the data center network relying on SDN.

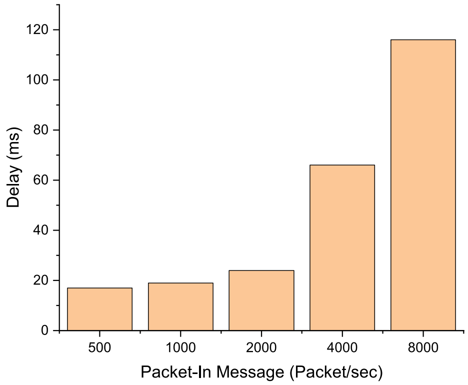


FIGURE 1. **The processing delay of Packet-In messages in an OpenDaylight SDN controller.**

In addition, the SDN-based failover mechanism in data center networks draws more researchers' attention. For instance, Li *et al.* [25] proposed a scalable failover method using OpenFlow. In their method, only three switches' flow table modifications are involved to handle a single link failure. A fast failure recovery method in load-balanced SDN-based data center networks is also suggested in [26], where an active probe mechanism is used to detect and manage failures. Jin *et al.* [27] focuses on virtualized SDN environments for clouds, proposing FAVE, which provides seamless failover and bandwidth-aware protection by allocating backup routes carefully. Also, energy consumption is fundamental to cloud-based data centers. As mentioned in [28], switches are the most energy consumer that should be controlled. Since the SDN technology enables the ability of allocating resource by a need, it's worth studying to reduce energy consumption in DCN using SDN technology, e.g. [29], [30] and [31].

Although software-defined networking outperforms conventional distributed routing algorithms in many aspects, the deployment of SDN in large scale data center networks is still immature. As mentioned in [25], data center networks have a high demand for scalability. By contrast, the limitation in the scalability of centralized control planes in SDN is obvious. More than thousands of network devices produce considerable pressure on OpenFlow controllers, which have a strong probability to be the bottleneck in DCN. Besides, the most common method of topology discovery, LLDP, tends to be extremely low-efficient in large scale topology. According to [32], on the widely-used SDN controller (OpenDaylight), the packet processing delay increases with the growing of Packet-In messages. As shown in Fig. 1, the processing delay in OpenDaylight has exceeded 100ms when the speed

TABLE 1. Comparison between cRetor and other routing schemes.

Scheme	cRetor	Fat-Tree / BCube / DCell	Conventional SDN	OSPF
Path caculation performance	High, A* based algorithm	N/A	Medium, Dijkstra	Medium, Dijkstra
Convergence	Fast	Fast	Medium	Medium
End-to-end delay	Medium, flow entries distributed at the first packet in a flow	Low	High, flow entries distributed at the first packet in a flow	Low
Protocol overhead	Medium	Low	High	High
Failover	Fast and almost no packet loss	Relatively slow	Slow	Fast with fast convergence enabled, slow otherwise
Scalability	High, TPDL is adaptive for large scale topologies; controller workload is much lower than conventional SDN	Very high	Very low, controller tends to be the bottleneck	Low, convergence slows down with large scale topologies
Generality	Medium, mainly for all kinds of regular network topology, topology structure required in advance	Low, only for Fat-Tree topology	High, for any topology	High, for any topology
Manageability	High, compatible with all kinds of SDN applications	Low	High, global view in controllers; direct control to all switches	Low

of Packet-In messages reaches 8000 packet/second. Although it is periodic, the cost brought by this topology detection mechanism is considerable. Furthermore, frequent topology discovery is unnecessary in a less-changed and low-failure-rate network like DCN. Therefore, further study on the practical deployment of SDN in data centers is required.

In view of all that has been mentioned so far, one may suppose that the combination of both efficiency from topology-aware routing methods and flexibility from SDN seems to be a new opportunity for high efficiency, scalability and manageability in data center networks. There are two common solutions when it comes to the blend of SDN and topology-aware routing methods: 1) the controller-side topology-aware mode and 2) both-sides topology-aware mode. The controller-side mode means that topology-aware methods are only applied in the controllers while SDN switches remain unchanged or little changed. In this mode, one benefits from the topology-aware methods without spending much on upgrading switches (which usually means hardware redesign and replacement). In contrast, the both-side mode denotes a complete reform on both controllers and switches for extreme performance. A switch equipped with a topology-aware processor forwards packets efficiently in a fault-free network even without the support of controllers. Accordingly, the topology-aware controllers only play a role when failures occur, which critically improves their scalability. In the present study, we focus on the controller-side mode.

In this paper, we propose an SDN-based topology-aware routing approach for large-scale data center networks: cRetor. This scheme takes advantage of the structure character for efficient topology discovery and path calculation in a regular network topology. As a result, the controllers' burden will be significantly relieved and are able to support more switches.

The comparison among cRetor and other routing algorithms are shown in Table 1. Benefiting from the distance formula in TPDL, the path calculation performance of cRetor is superior to traditional Dijkstra-based routing algorithms. In addition, as a more general topology-aware routing algorithm, cRetor's performance in convergence, routing protocol overhead, and scalability are slightly lower than dedicated routing algorithms like Fat-Tree two level routing and BCube source routing algorithms and DCell. However, it also has more advantages than general routing algorithms such as SDN and OSPF. In terms of generality, cRetor can be applied to any regular network topology including Fat-Tree and BCube. While in the aspect of manageability, cRetor fully inherits the advantages of SDN technology and is fully compatible with the upper-layer applications of SDN. Also, cRetor inherits some drawbacks of SDN, such as the relatively higher first-packet end-to-end delay. In a nutshell, cRetor is a tradeoff between the traditional general routing algorithm and the dedicated topology-aware routing algorithms, which tries to find a balance between efficiency and generality.

The main contributions of this article are as follows:

- A formal method is proposed to define and describe a regular network topology using an undirected graph of multi-type nodes. The concept of distance formula is also proposed for diminishing the overhead of routing path calculation. Furthermore, a topology description language TPDL is created for network designers to sketch data center networks especially networks with a regular topology.

- In cRetor, an A-star-based path calculation method is used to speed up the routing calculation. This heuristic method is adaptive to a variety of regular network topologies, utilizing the regularities of network topologies to speed up the routing calculation.
- We introduce two components in cRetor to take the place of the lower-efficient LLDP-based topology detection mechanism. The function of topology detection is accomplished by TPDL, which provides the controller with an initial topology. While the other feature of LLDP, fault detection, is replaced by a proactive failure reporting manner.

We implemented cRetor and evaluated it on the Mininet simulation platform. The experimental results show that cRetor has obvious advantages compared with the traditional OSPF algorithm and the existing SDN routing method as well as a DCell implementation in terms of path computation performance, network convergence speed, routing overheads and fault recovery capability.

The rest of this paper is structured as follows: section II presents the formal description method and TPDL; section III elaborates the architecture and algorithms in cRetor; section IV focuses on the experiments and results about cRetor compared with OSPF, Floodlight and DCell; section V explores related work; finally, section VI concludes the paper and mentions directions for future work.

II. DEFINITION AND DESCRIPTION OF REGULAR TOPOLOGY

As mentioned above, a data center network topology is usually regular and can be described either recursively or iteratively. Or rather, the locations, addressing and connections among nodes in data center networks have some regularities. To make full use of them, a well-defined description method is the first step.

In this section, a formalized description method of regular network topologies is presented. With this method, it is more explicit to illustrate the regularities of topology in a formalized way. Additionally, a corresponding domain-specific language TPDL is also designed to obtain more intuitive and parseable forms of topology description, which builds a bridge between formalized formulas and routing programs.

A. FORMALIZED DEFINITION OF REGULAR TOPOLOGY

As we know that a computer network is composed of network devices (including switches, routers, servers, etc.) and links among them. In the view of graph theory, an ordinary network could be regarded as an undirected graph composed of nodes and edges, which is described by

$$G = (V, E) \tag{1}$$

where V denotes a collection of nodes, i.e., a collection of network devices, and E represents a collection of edges, i.e., a collection of links among network devices.

However, further information is needed to demonstrate the structure property of regular topologies. If the nodes are divided into groups where nodes in the same group share similar patterns, one will be capable of clarifying these patterns in formal symbols. In this way, the regularities are embodied by an undirected graph with multiple types of nodes, which is expressed as follows:

$$G = \left(\cup_{t=1}^{k} V_t, \cup_{t=1}^{k} \cup_{t'=t}^{k} E_{tt'}\right) \tag{2}$$

This equation means all nodes in the network can be divided into k different sets, and the nodes in each set are similar in respect to their locations and/or connections. Similar to (1), a multi-type undirected graph is also composed of a set of nodes and a set of edges. In (2), $\cup_{t=1}^{k} V_t$ represents all nodes in the entire network, where each V_t is a collection of nodes in the same type. For instance, in a Fat-tree network, V_1 can be defined as the core switches group, while V_2 is the aggregation switches. $\cup_{t=1}^{k} \cup_{t'=t}^{k} E_{tt'}$ is a set of various edges, and each $E_{tt'}$ is a set of links between two types of nodes V_t and $V_{t'}$. Since there may not be physical connection between certain types of nodes, $E_{tt'}$ can be an empty set Φ.

Taking the typical 4-pods Fat-tree network topology in Fig. 2 for an example, the nodes in the topology can be divided into four groups: 1) core switches, 2) aggregation switches, 3) edge switches and 4) servers. Therefore, the k in (2) is set to 4.

According to the structure of Fat-tree topologies, every core switch connects to all pods. Moreover, all core switches are divided into $\frac{pod}{2}$ groups and the core switches in the same group connect to the same switch in every pod. In this case, each core switch is encoded by $V_1(x, y) : x$ is the group identification and y is the index in its group. Similarly, aggregation switches are also presented by $V_2(m, n)$, where m is the pod number this switch belongs to and n is the index in this pod. Now that V_1 and V_2 is defined, one can give the definition of E_{12}:

$$E_{12} = \{< V_1(x, y), V_2(m, n) > |x = n\}$$

It means E_{12} is a link set connecting V_1 and V_2 nodes only when the x attribute of V_1 node equals to the n attribute of V_2 node, i.e., the core switches in the group x connect all the aggregation switches whose indexes in their pod are equal to x.

As for edge switches, they are very similar to aggregation switches, and are defined as $V_3(p, q)$ (p for pod and q for index). The links between aggregation switches and edge switches are even simpler: they connect to each other inside a pod. So the following equation indicates the links between edge switches and aggregation switches:

$$E_{23} = \{< V_2(m, n), V_3(p, q) > |m = p\}$$

At last, the servers in this topology are connected to edge switches, and each switches has $\frac{pod}{2}$ servers

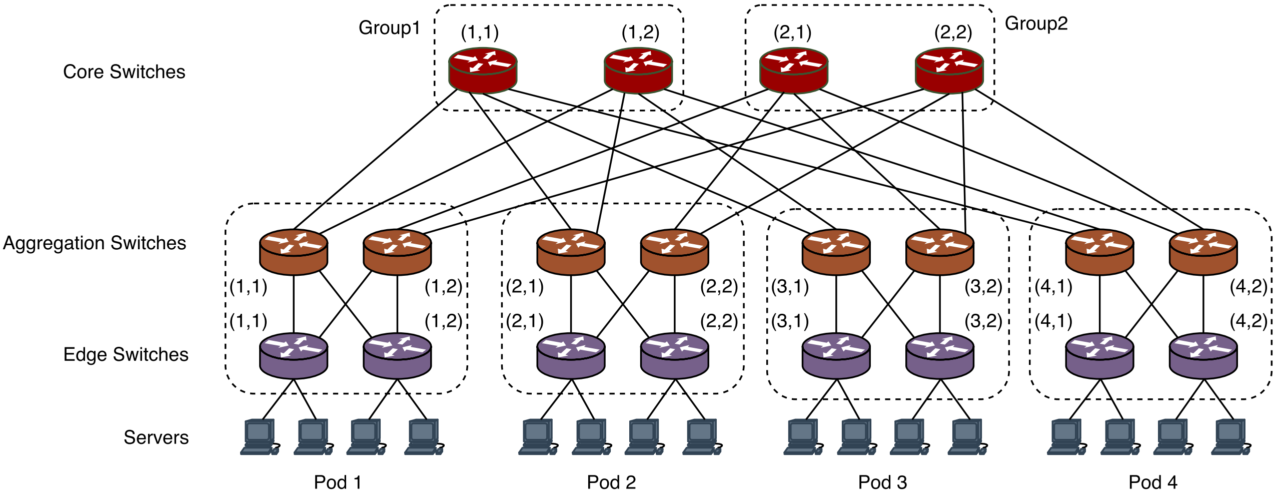


FIGURE 2. A 4-pods Fat-tree topology where nodes can be divided into four categories: 1) core switches, 2) aggregation switches, 3) edge switches and 4) servers.

attached.Therefore, $V_4(r, u, w)$ is used to indicate servers, where r is pod, u is the corresponding edge switch's index and w is the index in this subnet. Consequently, E_{34} will be $\{< V_3(p, q), V_4(r, u, w) > |p = r, q = u\}$.

In this topology, there is no respective internal connection in each node groups, so E_{11}, E_{22}, E_{33} and E_{44} are set to Φ. In the same manner, E_{13}, E_{14}, E_{24}, are also Φ.

In summary, the formalized description of a 4-pods Fat-tree topology is given below:

$$\begin{aligned}
G &= \left(\cup_{t=1}^{k} V_t, \cup_{t=1}^{k} \cup_{t'=t}^{k} E_{tt'}\right), k = 4, pod = 4\\
V_1 &= \{V(x, y)|1 \le x \le \frac{pod}{2}, 1 \le y \le \frac{pod}{2}\}\\
V_2 &= \{V(m, n)|1 \le m \le pod, 1 \le n \le \frac{pod}{2}\}\\
V_3 &= \{V(p, q)|1 \le p \le pod, 1 \le q \le \frac{pod}{2}\}\\
V_4 &= \{V(r, u, w)|1 \le r \le pod, 1 \le u \le \frac{pod}{2},\\
&\qquad 1 \le w \le \frac{pod}{2}\}\\
E_{12} &= \{< V_1(x, y), V_2(m, n) > |x = n\}\\
E_{23} &= \{< V_2(m, n), V_3(p, q) > |m = p\}\\
E_{34} &= \{< V_3(p, q), V_4(r, u, w) > |p = r, q = u\}\\
E_{11} &= \Phi, E_{22} = \Phi, E_{33} = \Phi, E_{44} = \Phi,\\
E_{13} &= \Phi, E_{14} = \Phi, E_{24} = \Phi \qquad (3)
\end{aligned}$$

where V_1 is the core switches set, V_2 is the aggregation switches set, V_3 is the edge switches set and V_4 represents the set of servers.

B. DISTANCE FORMULAS

A distance formula refers to an inductive form of distance (usually hops) between any two devices in the entire topology. Similar to the connections and locations, the distances in regular topologies also follow the same pattern. It will be beneficial for designing and implementing more efficient routing methods if we can describe the regularities explicitly.

Distance formulas and nodes connections are actually equivalent, but distance formulas are more intuitive to express the regularities and easier to be leveraged by routing algorithms. More specifically, the distance formulas can be thought of as a set of rules, each of which defines the distance of any two types of nodes in the topology under certain conditions. A distance formula can be expressed using the following quadruples:

$$< type_{src}, type_{dst}, condition, distance > \qquad (4)$$

It means that when the source node and destination node respectively belong to $type_{src}$ and $type_{dst}$, we will check whether the attributes of them satisfy the *condition*. If yes, the distance between the source node and the destination node is supposed to be *distance*.

For example, for the distance between servers located in different Fat-Tree network pods, one could use the following distance formula:

$$< s1 \in server, s2 \in server, s1.pod \mathrel{!}= s2.pod, 6 > \qquad (5)$$

It demonstrates that if the source and destination nodes are both server and their attributes r are not the same, the distance between them is 6 (hops).

Given any two nodes, the distance between them can be immediately obtained from distance formulas. Moreover, since the topologies are regular, it is ensured that a well-defined set of distance formulas won't increase rapidly as the network scale expands.

C. TOPOLOGY DESCRIPTION LANGUAGE TPDL

The formalized method of describing regular topologies has been introduced, but it's not sufficient in practical applications. On the one hand, the formalized description lacks some significant information for routing like IP addresses.

On the other hand, it is very difficult for computer programs to analyze the formalized description. Therefore, we proposed TPDL, a declarative domain-specific language.

TPDL involves not only all the components of formalized description method(like nodes, connections and distance formulas), but also addressing pattern and other information. Similar to the formalized description method, TPDL mainly consists of three parts: 1) network devices definitions, 2) links definitions and 3) distance formulas definitions.

1) DEVICES DEFINITIONS

Network devices such as switches and servers are the main components of a data center network. In the TPDL, the network devices are defined in groups by *device* blocks. Each *device* block is used to define a set of devices in the same type.

```
device AggSwitch {
    num: 8
    port: 4
    address: 0xC0000000
    attrs: {
        pod   = [1..4], 0x00FF0000
        index = [1..2], 0x000000FF
    }
}
```

This is a device block for aggregate switches in a Fat-tree topology. The *device* keyword indicates that the block is a device block, where a group of devices named *AggSwitch* is defined. The *address* keyword part is the base IP address of these nodes. The *attrs* keyword defines the custom device attributes *pod* and *index* of the device group. The last two hexadecimal digits are the mask of this attribute relative to the IP address. A mask is defined to indicate the mapping between the devices' attributes and IP addresses. For example, the attribute *index* indicates that if a device belongs to *AggSwitch* group, the value of attribute *index* will be the last eight bits of its IP address. In reverse, (6) is able to calculate the IP address of a device with k custom device attributes, where *tzn* is a function for counting the number of trailing zeros of the mask.

$$Addr = address + \Sigma_{i=1}^{k}[attr_i.value \ll tzn(attr_i.mask)] \quad (6)$$

2) CONNECTIONS DEFINITIONS

A link block is the minimum unit of links definition, and each link block contains a simple connection or a loop link definition. Different from the complicated connections in general networks, the connections in regular network topologies can be defined iteratively. Loop definition is supported by TPDL to define network connections to reduce the complexity of links definitions. Connections in a Fat-tree topology can be expressed by only 3 link blocks in TPDL. We use the <- -> symbol to define the connection between devices.

```
link {
    server[4] <--> server[7]
}
```

The link block above shows one of the simplest connection definitions. The *server* is a previously defined device type, and the values in brackets are the values of the custom device attributes. If there are multiple attributes, they should be list in the defined order. This link block connects two nodes in the *server* device group with custom device attribute values of 4 and 7.

```
link: {
    for i = 1..2,j = 1..4,z = 2..3 {
        EdgeSwitch[${j}][${i}] <-->
            server[${j}][${i}][${z}]
    }
}
```

This is a link definition with a loop and variables. Variables in TPDL are identified with *${var_name}*, where *var_name* is the name of the variable. There are two types of variables currently supported in TPDL: (1) device variable *${device_id.attr_name}* such as number of devices, number of device ports, etc.; (2) loop variables, which is defined in the loop statement.

Loop statements are mainly composed of loop variable definition statements and link statements. The loop variable is defined like this:

```
var_name = start..end[step]
```

Where *start* and *end* represent the start and end values of the loop variable. *Step* indicates the step size, which can be omitted when its value is 1. Loop variables can be defined one or more, and loop variables are separated by a comma. In TDL, multiple loop variables in a loop mean nested loops.

3) DISTANCE FORMULAS DEFINITIONS

In distance formulas blocks, devices' custom attributes are supported in the condition so that the distance formulas can express complex conditions.

```
distance server:s1, server:s2 {
    // s1 and s2 are in the same edge switch
    condition: s1.pod == s2.pod &&
             s1.edge == s2.edge => value: 2;

    // s1 and s2 are in the same pod but
    // different edge switches
    condition: s1.pod == s2.pod &&
             s1.edge != s2.edge => value: 4

    // s1 and s2 are in different pod
    condition: s1.pod != s2.pod => value: 6;
}
```

To define the distance formulas in TPDL, we first specify the node type of source node and destination node. In the example above, *s1* and *s2* are identifiers referring to devices of type *server*. The *condition* keyword defines a boolean expression that is expected to be *True*. Therefore, the first entry in the distance formula above means: when *s1* and *s2* are in the same pod and the same side, the distance will be 2.

When the distance between any two nodes is required, we just look up in the distance formula rules for a matched rule. In a regular network topology, the distance formula rules are usually not particularly large. For example, only 21 rules are needed to cover all cases for describing a standard Fat-Tree network.

In summary, TPDL brings together node definitions, connection definitions, and distance formulas in the data center network to provide a global view of the entire network topology for the control plane. In our current work, the TPDL is generated manually by network designers on the basis of analyzing the regularity of network topologies. But thanks to the regularity of data center networks, the size of the TPDL file is fairly controllable even in a large-scale data center network. In our experiments, we are able to describe a Fat-Tree network topology only by a 140-line TPDL file. More importantly, when a 16-server topology is extending to the scale of 1024 servers, there is only some modifications of parameters, but no new line is added. Therefore, we believe that TPDL is a simple but efficient way to describe regular network topology. In addition, we are also looking for a more convenient way to automatically generate distance formulas using techniques such as machine learning.

III. DESIGN AND ALGORITHMS

In this chapter, we elaborate on the architecture of the TPDL-based routing scheme cRetor, as well as the path calculation algorithm and failover mechanism in cRetor. At first, the basic framework is introduced, followed by the architecture of controllers and switches in cRetor. Besides, we present the path calculation algorithm based on A-star and TPDL's distance formula. Finally, the fault handling mechanism in cRetor is shown, including the detection and response to link failures.

A. FRAMEWORK OF cRetor

CRetor is a TPDL-based SDN routing framework for data center networks. It focuses on reducing the overheads of topology discovery and providing a more efficient route selection scheme to replace the traditional shortest path first (Dijkstra's) algorithm.

One of the key ideas in cRetor is to use TPDL as prior knowledge. By leveraging the information in TPDL, a cRetor controller can build a basic environment for ensuring that the whole network works. Other components are attached to enable cRetor to handle topology changes and failures. It is by nature that data center networks are more reliable and less changeable. Considering these features, in cRetor, most of the static and less changeable topology information is provided before system running and a small amount of variable information like link failures is obtained during runtime. Compared to the common LLDP topology detection mechanism in SDN, the overall system overheads are significantly reduced.

Fig. 3 shows the architecture of a controller in cRetor. The TPDL parser is responsible for parsing the input TPDL file and sending the parsed result to the Topology Manager. The Topology Manager will build a topology in memory according to the TPDL data and update it while the controller is running to keep it consistent with the real state of the networks. A distance-formula-based A-star algorithm is implemented in the Routing Calculator, where paths are selected as its name indicated. When a Packet-In message is received from the OpenFlow Module, the Routing Calculator finds an optimal path for this flow. The Fault Processor then notifies the Topology Manager of topology changes immediately when it gets changes from the OpenFlow Module.

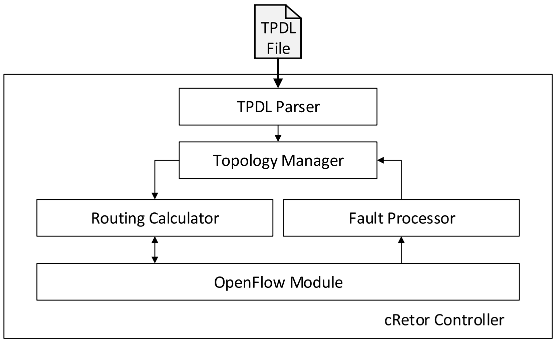


FIGURE 3. The architecture of cRetor controllers.

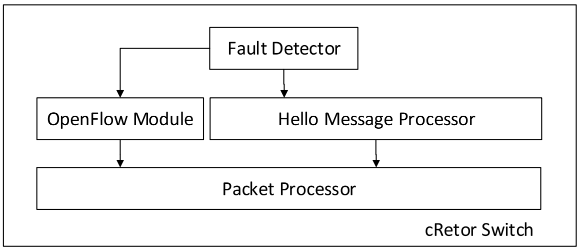


FIGURE 4. cRetor Switch Architecture.

The architecture of a cRetor switch is shown in Fig. 4. A Fault Detector (FD) module is added to the general OpenFlow switches, which will find the failures between the switches and their neighbors with the help of the Hello Message Processor (HMP). The HMP broadcasts Hello message to all connected ports periodically, where its own identifier is involved. Also, all received Hello packets will be forwarded to HMP to keep a record of the neighbors' information. The FD tracks the Hello messages from its neighbors to determine wether a fault occurs between the neighbor node and itself. When it doesn't receive a Hello message after a preset interval, it will send a fault report message to the controller via the OpenFlow Module. The complete mechanism of fault detection is discussed in subsection III-C.

In our implementation, we modified the standard Open vSwitch switches [33] into cRetor switches according to the proposed architecture. The controllers and switches in cRetor is compatible with conventional SDN implementaions, which means cRetor is able to inherit all the existing algorithms and infrastructures of SDN. For example, the traffic engineering methods mentioned in [34] or QoS Algorithms mentioned in [35] are still able to work on cRetor with minor modifications and they are supposed to benefit from the topology knowledge in cRetor.

B. PATH CALCULATION ALGORITHM

In most traditional SDN controllers, the routing selection relies on the SPF or CSPF algorithms. A controller obtains current network topology through the LLDP-based topology detection mechanism. Each time the network topology changes, the controller recalculates one or more shortest path(s) between nodes. Yen's k-shortest algorithm is adopted by most SDN controllers such as Floodlight, where Dijkstra's algorithm acts as the shortest path algorithm.

In our system, the controller does not need to perform topology discovery, because it knows the whole network topology from TPDL file. An initial network topology can be built from a TPDL file at startup on the controller, which is called Basic Topology. On the basis of Basic Topology, the controller maintains a latest topology, which is updated in real time according to the fault information reported by the switches.

Besides, as distance formulas are included in TPDL, we can get the distance between any two nodes in the network at low cost, which greatly improves the efficiency of the path calculating algorithm. Therefore instead of the Dijkstra's algorithm, a distance-formula-based A-star algorithm, where distance formulas act as a heuristic function of the A-star algorithm, is adopted in our scheme as the shortest path calculation algorithms. The A-star algorithm introduces a guess function, which provides a guess for the cost of the shortest path from current node to the destination node. The guess values are required to be lower than or equal to the real coast value to ensure that the algorithm will find the optimal path. The A-star algorithm with an exact guess function will directly traverse the shortest path to the destination. While an A-star algorithm with a guess function of zero corresponds to the original Dijkstra algorithm [35].

The key of the A-star algorithm is the order of traverse, which is guided and determined by a cost function $f(n)$. The evaluation function is as follows:

$$f(n) = g(n) + h(n) \tag{7}$$

where $g(n)$ is the actual cost (i.e., the distance) from the source node to current node n. The heuristic function $h(n)$ is the TPDL distance function (as shown in Algorithm 1), which indicates the cost from node n to destination node. Therefore $f(n)$ gives an estimated cost from the source node to the destination node via the intermediate node n. The traverse will proceed in the direction of the minimum $f(n)$. In other words, the more accurate the $h(n)$ is, the faster we will find the shortest path.

The combination of the A-star algorithm and distance formulas is supposed to work efficiently in both failure-free and partially failed network topologies. We use $d(n)$ to indicate the actual distance from node n to the destination node, and the details are as follows.

1) FAILURE-FREE NETWORK TOPOLOGIES

In a failure-free network, the distance formulas are functions that reflect the actual distance, which means that it is the real cost function, i.e., $h(n) = d(n)$. As a result, it is the most efficient heuristic function that will guide the A-star algorithm to find the shortest path in optimal time.

Algorithm 1 TPDL Distance Algorithm

```
Input: $n_s$: source node; $n_d$: destination node; list: list of
    distance formulas in a TPDL file
Output: distance between node $n_s$ and $n_d$
for each rule $\in$ list do
    if types of $n_s$ and $n_d$ match rule's requirement then
        value $\leftarrow$ compute rule's condition expression with $n_s$ and $n_d$
        if value = true then
            distance $\leftarrow$ rule.distance
            return distance
        end if
    end if
end for
distance $\leftarrow \infty$
return distance
```

2) NETWORK TOPOLOGIES WITH FAILURESP

In a partially failed network, the actual distance between source and destination might be larger than shortest distance, i.e., $h(n) <= d(n)$. In this case, the A-star algorithm works more time but is able to find shortest path eventually. The failed devices in a data center network usually account for only a small fraction of all network components. Hence it is most likely that there are only one or two link failures on the original shortest path. With the guidance of the distance formulas, the A-star algorithms will reach the failure node along the optimal path. And then it backtraces to bypass certain failures. Finally it continues to search for the destination node in the fastest way after bypassing.

C. FAILOVER MECHANISM

In data center networks, the requirements for network availability and reliability are often higher than those of general networks. Therefore, fault detection and recovery in data center networks are always more significant. In the SDN technology, the LLDP enables the functions of both 1) topology discovery and 2) fault detection. However, as the TPDL has been in charge of the topology discovery efficiently in cRetor, a lightweight fault detection mechanism is expected.

Instead of using LLDP, we adopt the mechanism of periodically sending Hello packets to detect faults. Each switch node periodically broadcasts Hello messages to all of its ports. When a switch receives a Hello message from its neighbor, it is ensured that the link to the neighbor node works well. By default, we use 3 times the broadcast interval as the fault timeout period. That is, if a switch does not receive a message from a neighboring node for 3 consecutive intervals, it will mark the corresponding link as failed. Then the switch will

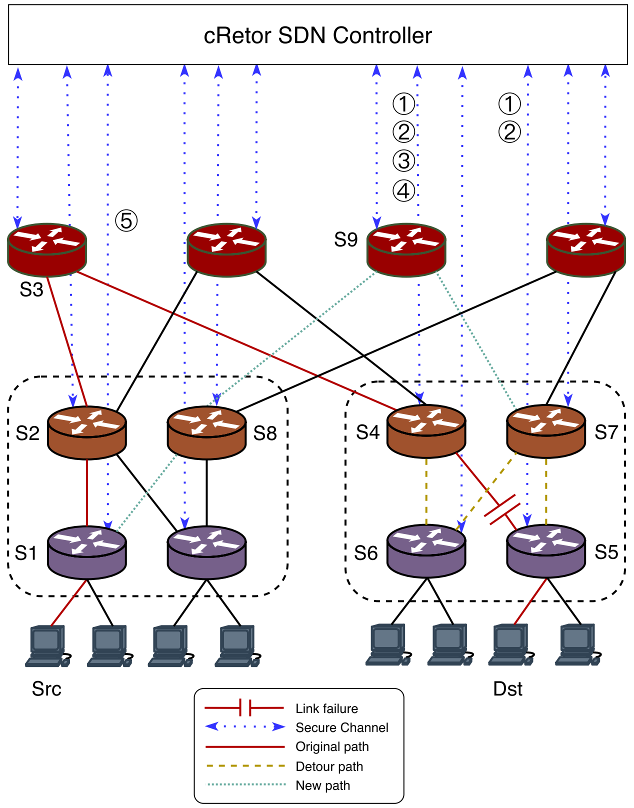


FIGURE 5. **A 4-pod Fat-Tree topology (only two pods are shown). The original path from Src to Dst is (Src, $S1$, $S2$, $S3$, $S4$, $S5$, Dst). After the link failure between $S4$ and $S5$ occurring, the detour path for packets on the way turns into ($S4$, $S6$, $S7$, $S5$). New path from Src to Dst becomes (Src, $S1$, $S8$, $S9$, $S7$, $S5$, Dst).**

reports the fault information to the controller through the secure channel.

A combination of proactive and reactive failover mechanism is designed for cRetor. When a failure occurs, the corresponding switch will report it to the controller by the Port-Status message. The controller will not only update its Current Topology but also proactively delete flow entries whose output port connects to the failed link from the corresponding switch. In this way, when a new packet arrives at this switch, a Packet-In message will be triggered because of the table-miss. After receiving the Packet-In message, the controller will run the A-star algorithm on the updated Current Topology to find a new path to avoid failures, and distribute flow entries to all the switches in the path. Every time the controller runs the A-star algorithms, it finds a path from the source node to the destination node of the packet instead of from the current switch to the destination node, so that the detoured paths can be avoided. The example given below reflects this mechanism.

In Fig. 5, a 4-pod Fat-Tree topology is given (only two pods are shown). Without failure, the original path of the flow from Src to Dst is (Src, $S1$, $S2$, $S3$, $S4$, $S5$, Dst). When the link failure between $S4$ and $S5$ occurs, the cRetor SDN controller processes the failure as follows:

① The link failure is detected by S4 and S5 via Hello messages. S4 and S5 will report this failure to the controller by Port-Status messages.
② The controller receives the failure information from $S4$ and $S5$ and updates its current topology. Then the controller distributes Flow-Mod messages to $S4$ and $S5$ to delete all flow entries whose outport is the failed port.
③ The packets from Src to Dst reach $S4$. Since the corresponding flow entry has been deleted, $S4$ sends a Packet-In message to the controller.
④ The controller calculates the shortest path from $S4$ to Dst, i.e. ($S4$, $S6$, $S7$, $S5$), and tell $S4$ to forward packets to $S6$.
⑤ The controller also calculates the shortest path from Src to Dst, and distributes flow entry to related switch $S1$, that the packets to Dst are supposed to be forwarded to $S8$ instead of $S2$ any more.

In this way, the new path from Src to Dst is modified to (Src, $S1$, $S8$, $S9$, $S7$, $S5$, Dst). The packets which have been forwarded to $S2$ in $S1$ will arrive at $S4$ following the original path, and then be redirected to detour path ($S4$, $S6$, $S7$, $S5$, Dst). After all packets in $S2$, $S3$ and $S4$ being processed, the data flow completely switches to the new shortest path.

IV. EXPERIMENTS AND EVALUATIONS

We compare cRetor with different kinds of routing schemes to demonstrate how cRetor performs. Firstly, OSPF is chosen as it's one of the most typical link-state routing algorithms and is broadly adopted in data center networks [36]. In addition, Floodlight is involved as the representative of conventional SDN, because cRetor is based on SDN and SDN is introduced to data center networks gradually. What's more, we adopt DCell topology and its routing algorithm since DCell is a typical topology-aware topology just like Fat-tree and BCube.

We implemented the TPDL parser using ANTLR [37] and integrated it into our cRetor controller, which is on the basis of open-source controller Ryu [38]. The cRetor controller and a modified Open vSwitch switch are used with our experiments to verify the feasibility and performance of cRetor. Several virtual Fat-Tree networks are built with different sizes using Mininet [39]. We also built standard SDN networks based on Floodlight and Open vSwitch by Mininet as well as OSPF networks using the Quagga [40] routing suite. In OSPF networks, we run multiple Quagga OSPF processes in different Linux network namespaces, which is also implemented by Mininet and very similar to the cRetor and SDN mode.

In addition, an SDN-based DCell implementation is adopted for comparison. As shown in [41], this implementation shares very similar results with the original experiment in DCell [7] in terms of fault tolerance and network capacity. DCell uses a different topology from Fat-tree in the aspects of topology structure and nodes number. However, DCell, as a very typical topology-aware routing algorithm, is also well-known like Fat-Tree and BCube.

The performance of cRetor is evaluated in the following aspects: 1) path calculation performance, 2) network convergence time, 3) control message overheads and 4) failure recovery time.

A. PATH CALCULATION PERFORMANCE

We first evaluate the performance of the controller's core path computation algorithm, using the A-star algorithm with distance formulas to compare with the commonly used Dijkstra's algorithm.

Different scale Fat-Tree network topologies are adopted for path computation performance testing, with scales of $k = 4$, $k = 8$, $k = 12$, and $k = 16$. The Dijkstra's algorithm and the A-star algorithm implementation in NetworkX [42] are adopted. In cRetor, a distance computation function is provided for the A-star algorithm in NetworkX as its heuristic function. The evaluation program runs on an Intel Core i7 3.41 GHz PC with Python 3.7 runtime environment.

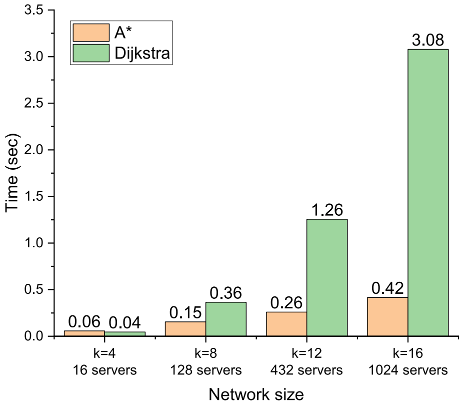


FIGURE 6. **Comparison of routing calculation time of the Dijkstra's algorithm and cRetor's A-star algorithm in different network scales.**

1000 pairs of source and destination nodes are chosen randomly as input parameters of the Dijkstra's and A-star algorithms. It is shown in Fig. 6 that when the network size is small, the calculation time of the two algorithms is very close, and the Dijkstra's algorithm even costs less time than A-star algorithm. With the increase of the Fat-Tree network sizes, it can be clearly seen that the time cost of the A-star algorithm with distance formulas is much less than the Dijkstra's algorithm. Furthermore, it is also illustrated in the figure that with the expansion of the network scale, the calculation time of Dijkstra's algorithm grows faster than that of the A-star algorithm. For a large-scale data center network, this near-linear growth rate is preferred.

The path calculation time in the Fat-Tree network ($k = 12$ and $k = 16$) with failures is also evaluated. As shown in Fig. 7, with the increase of link failure rate in the network, the time cost of the A-star algorithm with distance formulas growths. On the contrary, the time costs of the Dijkstra's algorithm tends to decrease. The reason is as follows, as the failure rate in the network increases, the errors in the prediction of the heuristic function will increase. Therefore the backtracking process needs to be performed more times and cost more time. While in Dijkstra's algorithm, higher link failure rate means less edges, and as a result, the calculation time reduces. It should be noted that in this experiment, even if the link failure rate has been as high as 20%, the time cost of A-star with distance formulas is still less than the Dijkstra's algorithm. In actual networks, especially the data center networks, it is almost impossible to find a scenario where there are 20% link failures at the same time.

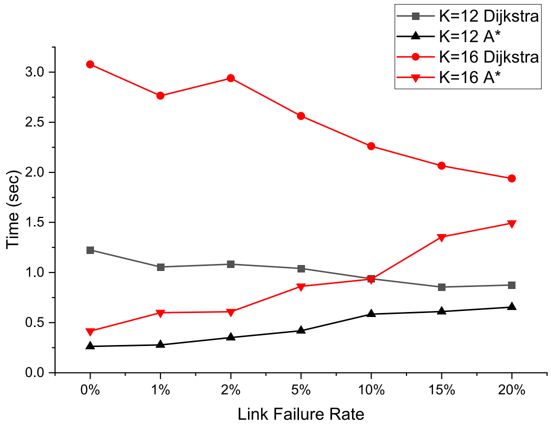


FIGURE 7. **Comparison of routing calculation time of the Dijkstra's algorithm and the cRetor's A-star algorithm in different link failure rates when $k = 12$ (432 servers in total) and $k = 16$ (1024 servers in total).**

B. NETWORK CONVERGENCE TIME

Secondly, we analyzed the network convergence time of cRetor and compared it with the LLDP-based SDN, traditional OSPF routing protocol and DCell. Mininet is utilized to build network topologies of different sizes for comparison of network convergence time. Because of the different topology structures, we use the closest network size for Fat-tree and DCell (16 servers and 128 servers for Fat-tree, 20 servers and 132 servers for DCell). The network convergence time is analyzed in the following four implementations: 1) Ryu-based cRetor controller and modified Open vSwitch switches, 2) Floodlight SDN controller and original Open vSwitch switches, 3) Quagga's OSPF routing algorithms and virtual Linux switches by Mininet and 4) DCell routing scheme based on POX controller and Open vSwitch switches.

Different convergence time measurement methods are chosen for cRetor, Floodlight and OSPF on the same principle: the time that the any two nodes are able to communicate with each other since the simulation starts. For the distributed OSPF routing algorithm, the time from the running of the network to the establishment of all the routing table entries in all switches is counted as the network convergence time. For the Floodlight SDN network, the time from the startup of the network to the time that the Floodlight controller detects all the links in the network through LLDP, i.e., the time when the Floodlight controller obtains the topology of the whole network, will be counted as its convergence time. For cRetor and DCell, as long as the switch establishes a connection with the controller, the network can be considered to have converged. Therefore, we choose two nodes in the network that are located in different Pods as the source and destination

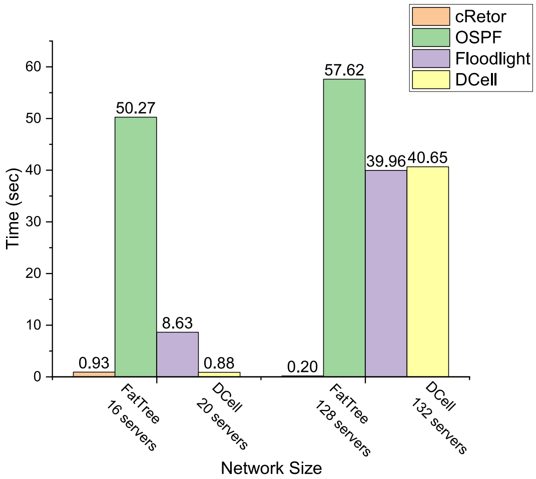


FIGURE 8. Comparison of network convergence time of cRetor, OSPF, Floodlight and DCell in different network sizes.

nodes. The time from network startup to the first packet reaches the destination node will be regarded as cRetor's convergence time.

It is demonstrated in Fig. 8 that in Fat-Tree networks of $k = 4$ and $k = 8$, the convergence time of cRetor is much smaller than the traditional link state protocol OSPF and LLDP-based SDN network. During the experiment, cRetor can complete the transmission of the first packet in less than 1 second. In comparison, both OSPF and Floodlight take tens of seconds to complete the detection and synchronization of the network topology. As for the DCell, in a small-sized network, it converges as long as the cRetor does. While in a larger network it takes much more time similar to the Floodlight.

The scalability of these three routing algorithms is also worth discussing from the convergence time with different scale of networks. For cRetor, when the network size is expanded from $k = 4$ to $k = 8$, there are only some fluctuations in the convergence time and basically no major changes occur. This is due to the fact that the main overhead in cRetor's convergence stage is the establishment of secure channels between switches and the controller, which is less performance-consuming. For OSPF, Floodlight and DCell, the convergence time increases as the network scale expands, because the exchanged information among switches and the controller (e.g. LSAs and DBs in OSPF and LLDP packets in Floodlights and DCell) will increase rapidly as the network scales up. Especially for Floodlight and DCell, the topology detection mechanism has pretty high-performance requirements for SDN controllers, hence the SDN controller is very likely to become a bottleneck in the entire network.

C. END-TO-END DELAY

In a typical SDN network, when the first packet of a flow is sent, the controller is supposed to find a forwarding path and establish it by distributing flow entries. Therefore, the delay of the first packet reflects the processing performance of the controller. While the delay of subsequent packets of this flow depends only on the network topology structure and the forwarding capability of switches. As a result, in the end-to-end delay evaluation, the delay of the first packet is selected as the evaluation criterion.

The Ping command is used to measure the first packet delay in Mininet network for both cRetor, Floodlight and DCell. We used different traffic models to evaluate the performance: One-to-One, One-to-All, and All-to-All, which are representative inter-data center traffic scenarios. The network for cRetor and Floodlight is a Fat-Tree topology with $k = 4$, that is, the numbers of flows in different traffic models are 1, 15 and 240, respectively. In DCell, a 20-server DCell network is used, with flow numbers of 1, 19 and 380. After convergence, the node(s) in the network will send out the first packet of the flows at the same time. The returned routing trip time (RTT) is divided by 2 to get the end-to-end delay. The results are as follows:

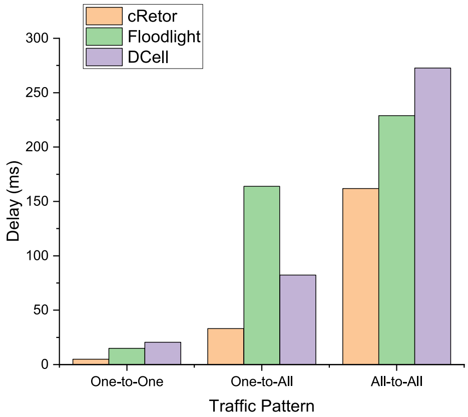


FIGURE 9. First packet end-to-end delay of cRetor and Floodlight (in a 16-server Fat-tree network), as well as DCell(in a 20-server DCell network).

We can find in Fig. 9 that the end-to-end delay in cRetor is lower than both Floodlight and DCell under all three different traffic modes, even they share the same SDN platform. One of the reasons is cRetor's LLDP-based A-star path calculation algorithm performs better than the Dijkstra algorithm in Floodlight. Also, the protocol overhead of cRetor is lower than the traditional SDN technology (As shown in the next evaluation). And the SDN version of DCell also inherits this topology discovery mechanism. Therefore, compared with conventional SDN solutions, cRetor performs better in first-packet end-to-end delay. Of course, the first packet delay of cRetor is still relatively high compared with Fat-Tree two-level routing, BCube source routing, and OSPF algorithms. This is because the forwarding decisions of these algorithms are determined on switches without interaction between switches and the controller. We leave this an open problem for future work to eliminate this kind of delay.

D. CONTROL MESSAGE OVERHEAD

We also use TCPDump to collect the control message overheads of the three routing algorithms for the first 120 seconds. The results are shown in Fig. 10 to Fig. 12. For OSPF, all the types of control messages specified in the OSPF spec

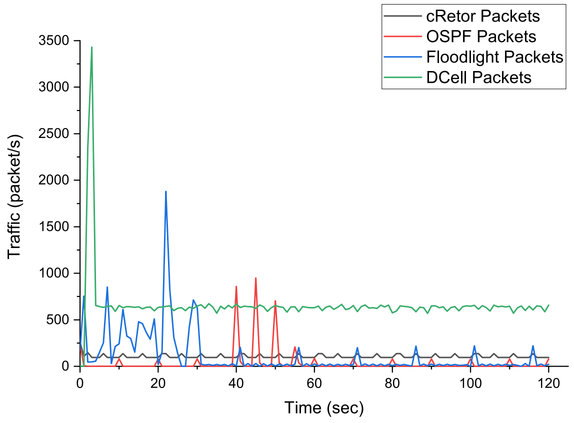


FIGURE 10. Control messages overheads in packet number of the 4 routing algorithms (16 servers for cRetor, OSPF and Floodlight, 20 servers for DCell).

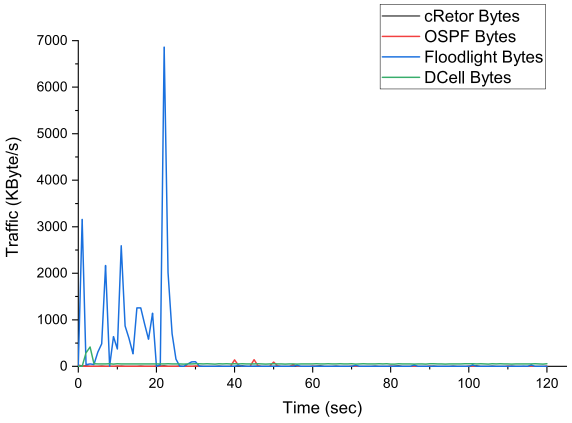


FIGURE 11. Control messages overheads in bytes of the 4 routing algorithms (16 servers for cRetor, OSPF and Floodlight, 20 servers for DCell).

are taken into account. Floodlight and DCell control packets include OpenFlow messages and LLDP messages between nodes. OpenFlow messages and Hello messages are counted for cRetor.

From the perspective of packets number in Fig. 10, the DCell sends the most packets both at convergence stage and stable stage with the highest speed of more 3000 packets/s. Although the server number of the DCell network is a little more than that of the Fat-tree network, the packet number is much higher than other routing algorithms. The reason is that it sets a smaller LLDP discovery interval (1 second) for a faster failure recovery time, which is smaller than that in Floodlight(15 seconds).

In OSPF and Floodlight, it takes tens of seconds for the control messages number to peak, which is corresponding to their convergence process (OSPF converges in about 50s and Floodlight converges in about 10s). After the convergence is complete, both OSPF and Floodlight are in a stable state. The corresponding topology detection packets (OSPF Hello for OSPF, Packet-In and Packet-Out of LLDP for FloodLight) are sent at regular intervals. As shown in Fig. 10, the Hello packet interval of OSPF is 10s and the LLDP detection interval of Floodlight is 15s. It should be noted that even our experimental scenario is a small Fat-tree topology with $k = 4$, the control packets reach 200 packets per flooding. For a k pod Fat-tree topology, the number of Packet-In messages will be:

$$N_{Packet-In} = N_{switch} \times N_{port} - N_{server} = k^3 \tag{8}$$

where N_{switch} is the number of all switches, N_{port} is port number per switch and N_{server} is the number of servers. Hence for a common $k = 48$ topology, the number of Packet-In message will be 110592, which is likely to lead to higher processing delay as mentioned in section I. It is also worth noting that there is an LSA update every 30 minutes which also raises many packets in OSPF.

For cRetor, the number of cRetor control messages fluctuates very little during the whole time, because there is no convergence process. Most of the control messages in cRetor are Hello messages. In the experiment, the hello message interval is set to 1 second, so the total number of Hello messages in cRetor is higher than the other two algorithms. However, unlike the LLDP detection packets in Floodlight, the Hello packets in the cRetor are among switches and won't be forwarded to the controller, so that there is no extra load to the controller. As a result, it consumes less CPU and network resources of switches and controllers.

From the perspective of bytes of control messages in Fig. 11, the total number of control messages of Floodlight during convergence is much larger than cRetor, OSPF and DCell. The data from 30s to 120s in Fig. 12 demonstrates the comparison among OSPF, cRetor and DCell more clearly. The traffic of DCell is still the largest even in the same topology discovery as cRetor. It also can be seen that the control message traffic in the OSPF convergence process is also much larger than the cRetor Hello packet traffic.

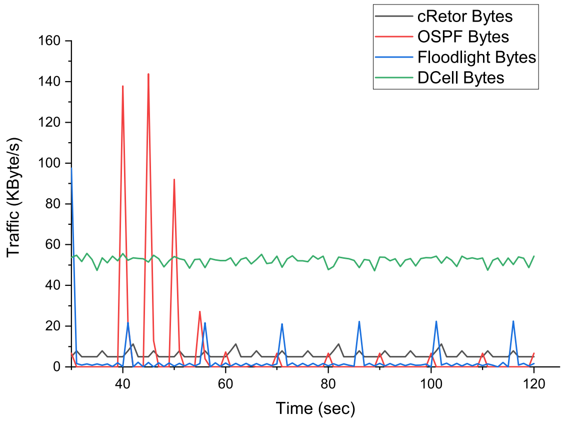


FIGURE 12. Control messages overheads in bytes of the 4 routing algorithms (16 servers for cRetor, OSPF and Floodlight, 20 servers for DCell).

Therefore, from the perspective of the control messages overhead, cRetor produces a relatively smooth and predictable control packet traffic and rarely has bursty control traffic. And the higher frequency of Hello packets enables cRetor to discover network failures faster.

E. FAILOVER TIME

Finally, we evaluated cRetor's failover capabilities and compared it to OSPF, Floodlight and DCell. A pair of hosts in different pods of the experiment network are selected as the source and destination nodes. On the source node, a client program runs, which sends a UDP packet to the server every 1 millisecond, with an incremental sequence number in it. On the destination node, a server program is running, which will listen to the specific UDP port. It will store the sequence numbers in received packets and interval between this packet and the last packet.

The other equivalent path from the source node to the destination node is disconnected in advance, leaving only one shortest path. In other words, the routing algorithms have to find a longer feasible path for new packets after the failure occurs.

In our experiments, data flow could not be recovered in Floodlight. It could be found that the default idle age of the flow table entries sent by Floodlight is 5 seconds, which means, if no packet hits this flow table entry within 5s, the flow table will be deleted automatically. In this way, Floodlight can handle situations where the link is down and the data flow stops as well. However, in our experiments, the UDP data flow never stops and the data packets will always hit the invalid flow table entry, causing the flow entry cannot be deleted and the data transmission cannot be resumed.

In the experiment for OSPF, the OSPF Fast Convergence feature has a great impact on the experimental results. In the case where Fast Convergence is not enabled by default, the delay of the SPF Timer is 5s, while the initial value of the SPF Timer is 50ms after that feature being turned on. According to Cisco's documentation [43], the Fast Convergence feature will turn on by default since Apr 2017 (IOS Release 16.5.1). So we conducted our experiments with both this feature on and off.

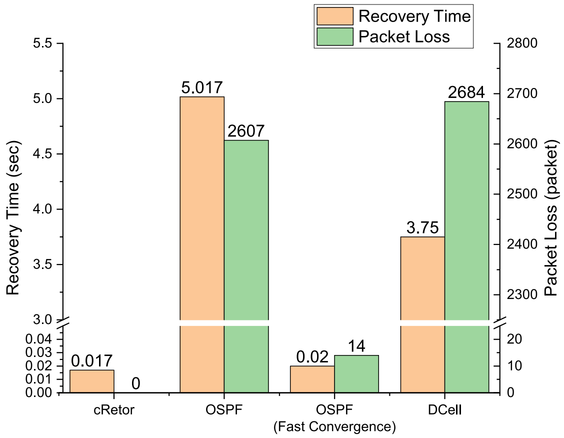


FIGURE 13. The recovery time and lost packet number of UDP flow. cRetor, OSPF and OSPF with fast convergence enable are in a 16-server Fat-tree network, while DCell is in a 20-server DCell network.

As shown in Fig. 13, the OSPF algorithm with Fast Convergence off has a flow recovery time of 5 seconds, during which more than 2,600 packets are lost. As for the DCell, it takes more than 3 seconds to reconfigure the switches to forward the flow again. And similar to the OSPF, more than 2600 packets are lost during reconfiguration. While for the OSPF with Fast Convergence enabled, the flow recovery time is greatly reduced, and the reconvergence is completed in about 20ms. But in the process, a small number of packets are still lost. The recovery time of cRetor outperformed the OSPF algorithm. The link switching delay is only 17ms, and no packet loss occurs during the handover.

V. RELATED WORK

The emergence of the SDN leads to the revolution of networking programmability from user configuration of routers and FPGA-based hardware programming to the new OpenFlow-based diagram with decoupled control and data plane as well as centralized controllers. This improvement of networking programmability makes the network more dynamic, robust and able to experiment with new ideas and protocols [44].

In recent years, programmability in the control plane of networks is moving from low-level languages such as OpenFlow to higher-level languages. High-level programming languages can be powerful tools for implementing and abstracting different important functions of SDN such as network-wide structures, distributed updates and virtualization [15]. NetCore [45] is a high-level, declarative language for defining packet-forwarding policies on SDNs. FatTire [46] focuses more on the degree of fault tolerance required, though it is also used to specify the forwarding rules of packets in the network. In contrast, TPDL pays more attention to the network topology itself instead of forwarding strategies for packets. TPDL itself is a tool for network topology description, which provides support for upper-layer network forwarding decisions.

The existing software-defined networking scheme enables the programmability of the networking control plane, while the forwarding process is still burned in the switch chip. As a result, there is increasing concern over the extensibility in new protocols and actions supporting. The updates of new protocols and features rely heavily on hardware redesigns by networking vendors, which also means higher cost and longer update period. Consequently, P4 [47] is proposed to allow one to program packet parsing and forwarding, setting more open networking and devices in motion. Instead of being created as a replacement of P4, TPDL works in the control planes, leveraging the data plane programmability brought by P4 to lay the foundation for more efficient forwarding schemes. Moreover, as mention in section I, TPDL parser could be embedded into switches for higher-performance. In that situation, one can make use of the capacities of P4 for implementing switch-side routing scheme, like storing distance formulas and neighbor table looking up.

VI. CONCLUSION AND FUTURE WORK

In this paper, an SDN-based topology-aware routing scheme cRetor for regular network topologies is proposed, which leverages the regularities of topology in the data center networks to achieve efficient topology description and distance calculation. A formalized topology description method and corresponding topology description language are created, so that network operators can describe the topology of the entire network in a simple way. Then we designed and implemented a routing scheme based on TPDL and SDN technology, as well as an efficient routing calculation method and fault handling mechanism based on the A-star algorithm and TPDL. The experimental results show that compared with the OSPF, the conventional SDN network with Floodlight controller and DCell, the route calculation of the cRetor is faster, the network convergence time is shorter, the control message overheads are more stable and predictable, and the fault recovery performance is also excellent. It is believed that cRetor can exploit the potential of data center networks better and improve the efficiency of the network.

Further research on TPDL and cRetor are also in process, such as 1) TPDL application in both controllers and switches and 2) automatic generation of distance formulas by machine learning algorithms. There are also many open problems in the framework of cRetor, e.g., 1) adaptability to network dynamic changes, 2) load balancing using the global view of SDN and the equal-cost multipath feature in data center networks, and 3) integrate additional information (such as bandwidth, etc.) into the distance formulas to get a more accurate cost.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, p. 63.

[2] S. Azizi, N. Hashemi, and A. Khonsari, "HHS: An efficient network topology for large-scale data centers," *J. Supercomput.*, vol. 72, no. 3, pp. 874–899, Jan. 2016.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, p. 51.

[4] M. Walraed-Sullivan, A. Vahdat, and K. Marzullo, "Aspen trees: Balancing data center fault tolerance, scalability and cost," in *Proc. 9th ACM Conf.*, New York, New York, USA, 2013, pp. 85–96.

[5] Y. Yu and C. Qian, "Space shuffle: A scalable, flexible, and high-performance data center network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3351–3365, Nov. 2016.

[6] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 63–74.

[7] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, p. 75.

[8] T. A. Nguyen, D. Min, E. Choi, and T. D. Tran, "Reliability and availability evaluation for cloud data center networks using hierarchical models," *IEEE Access*, vol. 7, pp. 9273–9313, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8604034/

[9] F. Gilabert, M. E. Gãmez, P. Lápez, and J. Duato, "On the influence of the selection function on the performance of fat-trees," in *Euro-Par Parallel Processing* (Lecture Notes in Computer Science), vol. 4128, W. E. Nagel, W. V. Walter, and W. Lehner, Eds. Berlin, Germany: Springer, 2006, pp. 864–873. [Online]. Available: http://link.springer.com/10.1007/11823285_91

[10] A. Farouk and H. M. El-Boghdadi, "On the influence of selection function on the performance of fat-trees under hot-spot traffic," in *Proc. 9th IEEE/ACS Int. Conf. Comput. Syst. Appl. (AICCSA)*, Dec. 2011, pp. 120–127. [Online]. Available: http://ieeexplore.ieee.org/document/6126622/

[11] J. Cao, D. Maltz, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, and Y. Xiong, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, 2013, pp. 49–60. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2535372.2535375

[12] Z. Qian, F. Fan, B. Hu, K. L. Yeung, and L. Li, "Global round robin: Efficient routing with cut-through switching in fat-tree data center networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2230–2241, Oct. 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8469071/

[13] A. Zhao, Z. Liu, J. Pan, and M. Liang, "A novel addressing and routing architecture for cloud-service datacenter networks," *IEEE Trans. Services Comput.*, early access, Oct. 8, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8862883/, doi: 10.1109/TSC.2019.2946164.

[14] S. Habib, F. S. Bokhari, and S. U. Khan, "Routing techniques in data center networks," in *Handbook Data Centers*. New York, NY, USA: Springer, Mar. 2015, pp. 507–532.

[15] J. Esch, "Prolog to, 'software-defined networking: A comprehensive survey,"' *Proc. IEEE*, vol. 103, no. 1, pp. 10–13, Jan. 2015.

[16] S. Jain, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, and J. Zhou, "B4: Experience with a globally-deployed software defined WAN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 1–10.

[17] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 74–87.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, Apr. 2010, pp. 9–19.

[19] Y. Afek, A. Bremler-Barr, S. L. Feibish, and L. Schiff, "Detecting heavy flows in the SDN match and action model," *Comput. Netw.*, vol. 136, pp. 1–12, Dec. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128618300859

[20] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 254–265.

[21] R. Trestian, G.-M. Muntean, and K. Katrinis, "Micetrap: Scalable traffic engineering of datacenter mice flows using openflow," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage.*, Mar. 2013, pp. 904–907.

[22] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 1581–1589.

[23] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks," in *Proc. Asia Pacific Signal Inf. Process. Assoc. Annu. Summit Conf.*, Feb. 2012, pp. 1–8.

[24] H. Zhang, F. Tang, and L. Barolli, "Efficient flow detection and scheduling for SDN-based big data centers," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 5, pp. 1915–1926, May 2019. http://link.springer.com/10.1007/s12652-018-0783-6

[25] J. Li, J. Hyun, J.-H. Yoo, S. Baik, and J. W.-K. Hong, "Scalable failover method for data center networks using OpenFlow," in *Proc. IEEE Netw. Oper. Manage. Symp. (NOMS)*, May 2014, pp. 1–6.

[26] B. Raeisi and A. Giorgetti, "Software-based fast failure recovery in load balanced SDN-based datacenter networks," in *Proc. 6th Int. Conf. Inf. Commun. Manage. (ICICM)*, Oct. 2016, pp. 95–99.

[27] H. Jin, G. Yang, B.-Y. Yu, and C. Yoo, "FAVE: Bandwidth-aware failover in virtualized SDN for clouds," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2019, pp. 505–507. [Online]. Available: https://ieeexplore.ieee.org/document/8814526/

[28] A. Shirmarz and A. Ghaffari, "Performance issues and solutions in SDN-based data center: A survey," *J. Supercomput.*, vol. 2020, pp. 1–49, Jan. 2020. [Online]. Available: http://link.springer.com/10.1007/s11227-020-03180-7

[29] S. Subbiah and V. Perumal, "Energy awake network traffic steering using SDN in cloud environment," in *Proc. 2nd Int. Conf. Recent Trends Challenges Comput. Models (ICRTCCM)*, 2017, pp. 31–36. [Online]. Available: http://ieeexplore.ieee.org/document/8057504/

[30] Q. Liao and Z. Wang, "Energy consumption optimization scheme of cloud data center based on SDN," *Procedia Comput. Sci.*, vol. 131, pp. 1318–1327, Oct. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050918307075

[31] M. D. S. Conterato, T. C. Ferreto, F. Rossi, W. D. S. Marques, and P. S. S. de Souza, "Reducing energy consumption in SDN-based data center networks through flow consolidation strategies," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, Apr. 2019, pp. 1384–1391. http://dl.acm.org/citation.cfm?doid=3297280.3297420

[32] S. I. Alliance. *Whitepaper on SDN Controller Performance in Data Center Scenario*. Accessed: Jan. 14, 2020. [Online]. Available: https://www.ixiacom.com/zh/resources/sdn-controller-performance

[33] *Open vswitch*. Accessed: Nov. 20, 2019. [Online]. Available: http://www.openvswitch.org/

[34] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 918–953, 2nd Quart., 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7762818/

[35] 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. [Online]. Available: http://ieeexplore.ieee.org/document/8027021/

[36] P. Zeng, Y. Shen, Z. Qiu, Z. Qiu, and M. Guo, "SRP: A routing protocol for data center networks," in *Proc. 16th Asia–Pacific Netw. Oper. Manage. Symp.*, Hsinchu, China, Sep. 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/6996564/

[37] *Antlr Another Tool for Language Recognition*. Accessed: Nov. 16, 2019. [Online]. Available: https://www.antlr.org/index.html

[38] *RYN SDN Framework*. Accessed: Nov. 16, 2019. [Online]. Available: https://osrg.github.io/ryu/

[39] *Mininet: An Instant Virtual Network on Your Laptop (or Other PC)*. Accessed: Nov. 16, 2019. [Online]. Available: http://mininet.org/

[40] *Quagga Routing Suite*. Accessed: Nov. 16, 2019. [Online]. Available: https://www.nongnu.org/quagga/index.html

[41] *Dcell Data Center Network Structure Implemented With Software-Defined Networking (SDN)*. Accessed: Apr. 20, 2020. [Online]. Available: https://github.com/chuyangliu/dcell

[42] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using network," in *Proc. 7th Python Sci. Conf.*, G. el Varoquaux, T. Vaught, and J. Millman, Eds, Pasadena, CA USA, 2008, pp. 11–15.

[43] L. D. Ghein. *Change of Default OSPF and is-is SPF and Flooding Timers and ISPF Removal*. Accessed: Nov. 20, 2019. [Online]. Available: https://www.cisco.com/c/en/us/support/docs/ip/ip-routing/211432-Change-of-Default-OSPF-and-IS-IS-SPF-and.html

[44] F. A. Lopes, M. Santos, R. Fidalgo, and S. Fernandes, "A software engineering perspective on SDN programmability," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1255–1272, 2nd Quart., 2016.

[45] C. Monsanto, N. Foster, R. Harrison, and D. Walker, "A compiler and run-time system for network programming languages.," in *Proc. POPL*, 2012, p. 217.

[46] M. Reitblatt, M. Canini, A. Guha, and N. Foster, "FatTire—Declarative fault tolerance for software-defined networks," in *Proc. HotSDN*, 2013, p. 109.

[47] P. Bosshart, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.

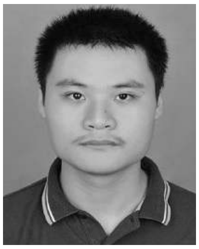
ZEQUN JIA received the B.E. degree in computer science and technology from Beijing Jiaotong University, China, where he is currently pursuing the Ph.D. degree with the School of Computer and Information Technology. His research interests include data center networking, software-defined networking, vehicular networking, and information-centric networking.

YANTAO SUN (Member, IEEE) received the B.S. degree from the Shandong University of Technology, in 1999, the M.S. degree from Shandong University, in 2002, and the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, in 2006. He studied at Columbia University as a Visiting Scholar, from September 2012 to September 2013. He is currently an Associate Professor with the School of Computer and Information Technology. He has published over 30 articles on international journals and conferences, such as WCMC, *Mobile Networks and Applications*, the *Journal of Communication*, GLOBECOM, LCN, and the *Journal of Software*. He also holds six patents. His research interests include cloud computing, data center networks, wireless sensor networks, the Internet of Things, multimedia communication, and network management.

QIANG LIU was born in Changsha, Hunan, China, in 1980. He received the B.S. and Ph.D. degrees in communication and information system from the Beijing Institute of Technology, Beijing, China, in 2002 and 2007, respectively.

From 2007 to 2018, he was an Assistant Professor with the School of Computer and Information Technology, Beijing Jiaotong University, where he has been an Associate Professor, since 2019. He is the author of more than 50 articles and four patents. His research interests include mobile ad-hoc networks, UAV ad-hoc networks, wireless media access control, wireless routing, and swarm intelligence.

SONG DAI received the B.E. degree in software engineering from Jiangxi Agricultural University. He is currently pursuing the M.E. degree with the School of Computer and Information Technology, Beijing Jiaotong University, China. His research interests include software-defined networking and mobile ad hoc networking.



CHENGXIN LIU received the B.E. degree in computer science and technology from Beijing Jiaotong University, China, where he is currently pursuing the M.E. degree with the School of Computer and Information Technology. His research interests include software-defined networking and satellite networking.