1-1-2016

Auto-Configuration of ACL Policy in Case of Topology Change in Hybrid SDN

Rashid Amin
COMSATS University Islamabad

Nadir Shah
COMSATS University Islamabad

Babar Shah
Zayed University

Omar Alfandi
Zayed University

Recommended Citation

Amin, Rashid; Shah, Nadir; Shah, Babar; and Alfandi, Omar, "Auto-Configuration of ACL Policy in Case of Topology Change in Hybrid SDN" (2016). *All Works*. 625.
https://zuscholars.zu.ac.ae/works/625

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 3, 2016; accepted December 14, 2016, date of publication December 19, 2016, date of current version January 27, 2017.

Digital Object Identifier 10.1109/ACCESS.2016.2641482

Auto-Configuration of ACL Policy in Case of Topology Change in Hybrid SDN

RASHID AMIN[1,2], NADIR SHAH[1], BABAR SHAH[3], AND OMAR ALFANDI[3]
[1]COMSATS Institute of Information Technology, Wah Cantt 47040, Pakistan
[2]University of Engineering and Technology, Taxila 47040, Pakistan
[3]Zayed University, Abu Dhabi 144534, United Arab Emirates

Corresponding author: N. Shah (nadirshah82@gmail.com)

This work is partially sponsored by Zayed University in the UAE under the Research Incentive Fund (RIF). Grant No. R15047.

ABSTRACT Software-defined networking (SDN) has emerged as a new network architecture, which decouples both the control and management planes from data plane at forwarding devices. However, SDN deployment is not widely adopted due to the budget constraints of organizations. This is because organizations are always reluctant to invest too much budget to establish a new network infrastructure from scratch. One feasible solution is to deploy a limited number of SDN-enabled devices along with traditional (legacy) network devices in the network of an organization by incrementally replacing traditional network by SDN, which is called hybrid SDN (Hybrid SDN) architecture. Network management and control in Hybrid SDN are vital tasks that require significant effort and resources. Manual handling of these tasks is error prone. Whenever network topology changes, network policies (e.g., access control list) configured at the interfaces of forwarding devices (switches/routers) may be violated. That creates severe security threats for the whole network and degrades the network performance. In this paper, we propose a new approach for Hybrid SDN that auto-detects the interfaces of forwarding devices and network policies that are affected due to change in network topology. In the proposed approach, we model network-wide policy and local policy at forwarding device using a three-tuple and a six-tuple, respectively. We compute graph to represent the topology of the network. By using graph difference technique, we detect a possible change in topology. In the case of topology change, we verify policy for updated topology by traversing tree using six-tuple. If there is any violation in policy implementation, then affected interfaces are indicated and policies that need to be configured are also indicated. Then, policies are configured on the updated topology according to specification in an improved way. Simulation results show that our proposed approach enhances the network efficiency in term of successful packet delivery ratio, the ratio of packets that violated the policy and normalized overhead.

INDEX TERMS Topology change, policy configuration, tree, graph difference, communication switching.

I. INTRODUCTION

Most recently, Software Defined Networking (SDN) has emerged as a new network architecture which decouples both the control and management planes from data plane at the forwarding devices. In SDN, the control and management planes are implemented at a central device which is called controller. The data plane is implemented at forwarding devices. Though SDN has many advantages over traditional networking, e.g. ease of both network management and enforcement of security policies in SDN [1]. However, SDN deployment is not widely adopted due to the budget constraints of organizations. This is because organizations are always reluctant to invest a large amount of budget in establishing a new network infrastructure from scratch [2]. One feasible solution is to deploy a limited number of SDN-enabled devices along with traditional (legacy) network devices in the network of an organization by incrementally replacing traditional network devices by SDN devices. This is called hybrid software defined networking (Hybrid SDN) architecture [25]. Both network policies and topology of network change frequently which cause network faults by creating network inconsistency and invariants in term of network policies [3]. In this case, Hybrid SDN requires manual configuration of legacy devices (switches and routers) by the network administrator.

Often, these faults get unnoticed for a longer time. It is reported that 62% network failures are due to human error and maintenance, and operations of such networks need 80% budget [4], [26]. As there is a central controller in hybrid SDN and this controller can have the overall topology view of the network, therefore it is possible that we can automatically configure the policies on both SDN-enabled devices and legacy devices [1].

The existing approaches, e.g. [5]–[7], do not deal with automatic policy configuration in case of a topology change in Hybrid SDN, to the best of our knowledge. We proposed an automatic policy enforcement mechanism, called Auto-PDTC, which both auto-detects policy violation in the case of network topology changes and enforces policy by configuring the policy on affected device's interfaces. Our contributions are

- For hybrid SDN, we identify the problem that network topology changes frequently by addition or removal of links/device. Due to this, network policies are violated at legacy devices in case of topology change. This problem is solved as follows.
- In Auto-PDTC, we model network-wide policy and local policy at forwarding device using a 3-tuple and a 6-tuple, respectively.
- Auto-PDTC gets the link state information from both legacy and SDN switches and then constructs the network topology in the form of a graph.
- Through graph matching, Auto-PDTC auto-detects the change in network topologies taken at different instances of times.
- In the case of topology change, Auto-PDTC verifies the policy for updated topology by traversing tree using 6-tuple. If there is any violation in policy implementation, then affected interfaces are indicated and policies that need to be configured are indicated.

The rest of the paper is organized as follows. Section II presents the problem statement. Related work is presented in Section III. The detail of proposed solution is explained in Section IV. In Section V simulation results are presented and Section VI concludes the paper.

II. PROBLEM STATEMENT

Link changes and the addition of new devices, these are the common events that occur frequently in the network and affect the network performance [3]. Several network policies, e.g. access control lists (ACL) [8], load balancing [8], etc., implemented in the network are badly affected in these situations. Because when the link changes, packets may violate the network policy (e.g. ACL) and subsequently network traffic flows to an unauthorized area. For example, there is an enterprise network for a company as shown in Figure 1(a). The company has two sites A and B at some different places. Site A has front offices where the computers (AF1 and AF2) are placed and these are connected to data centers (AD1 and AD2). Site B also has computers (BF1 and BF2) in front offices that are connected to data centers (BD1 and BD2).

Case 1: Suppose a company has ACL policy, say P1, that data centers of a site (say site A) can only be accessed from front office computers of site A. Otherwise, the data center of the site cannot be accessed from other places. More specifically AF1-AF2 can communicate to AD1-AD2 and BF1- BF2. Similarly, AF1-AF2 cannot communicate to BD1 and BD2. This policy is implemented using ACL commands on router's interface. Suppose, interface i2.1 is configured to drop all packets originated from BF1-BF2 subnet and interface i3.1 is configured to drop all packets originated from the AF1-AF2 subnet.

Case2: Later on, suppose it **is** decided by a network administrator to place a new link between R2 and R3 as shown in Figure 1(b) and the ACL policy is unchanged. Now the packets originated by AF1-AF2 subnet can reach to BD1-BD2 by passing through R1, R2, and R3 respectively as shown by arrows in Figure 1(b). In this case, ACL policy P1 installed on the interface i3.1 of R3 is bypassed and violated by allowing packets originated by AF1-AF2 subnet to reach BD1-BD2. In this case, it needs manual configuration by a network administrator to both detect this situation and configure the policy on both new interfaces that connects R2 and R3. To manually detect such policy violation, it is very hard in a large network and may be unnoticed for a long time [4]. Therefore, this requires an intelligent mechanism to automatically both detect this situation and to configure the new topology as per ACL policy.

Case 3: Suppose ACL policy violation is manually detected in Case 2. Then the network administrator installs P1 at interface i2.2 of R2 by discarding the packets originated from AF1-AF2 and at interface i3.2 of R3 by discarding the packets originated from BF1-BF2. This way of policy implementation is not a better option. In this case, suppose that if the link between R4 and F1 gets down as shown in Figure 1(c), then AF1-AF2 subnet cannot communicate with BF-BF2 subnet, though there is a path following R1, R2, R3 and R4 from AF1-AF2 to BF1-BF2, as this can be noticed in Figure 1(c). This is because the packet originated from AF1-AF2 will be discarded at interface i1.2 of R2 due to ACL policy P1 implementation. In this situation, an improved mechanism is needed for configuring/installing ACL policies at the interfaces of devices so that only affected interfaces are blocked and redundant path can be used in the network as shown in Figure 1(d).

These problems of ACL policy violation occur in hybrid SDN because SDN controller only controls the data flow through SDN switches [1]. In addition, the legacy devices use traditional network protocols to forward the data flows. In order to control legacy devices in hybrid SDN, customized algorithms are needed to be implemented at the SDN controller. More specifically, if data packets of a flow are passing through only legacy switches then packets are forwarded using traditional network protocols. In our targeted scenario, the packets pass through only legacy devices in all

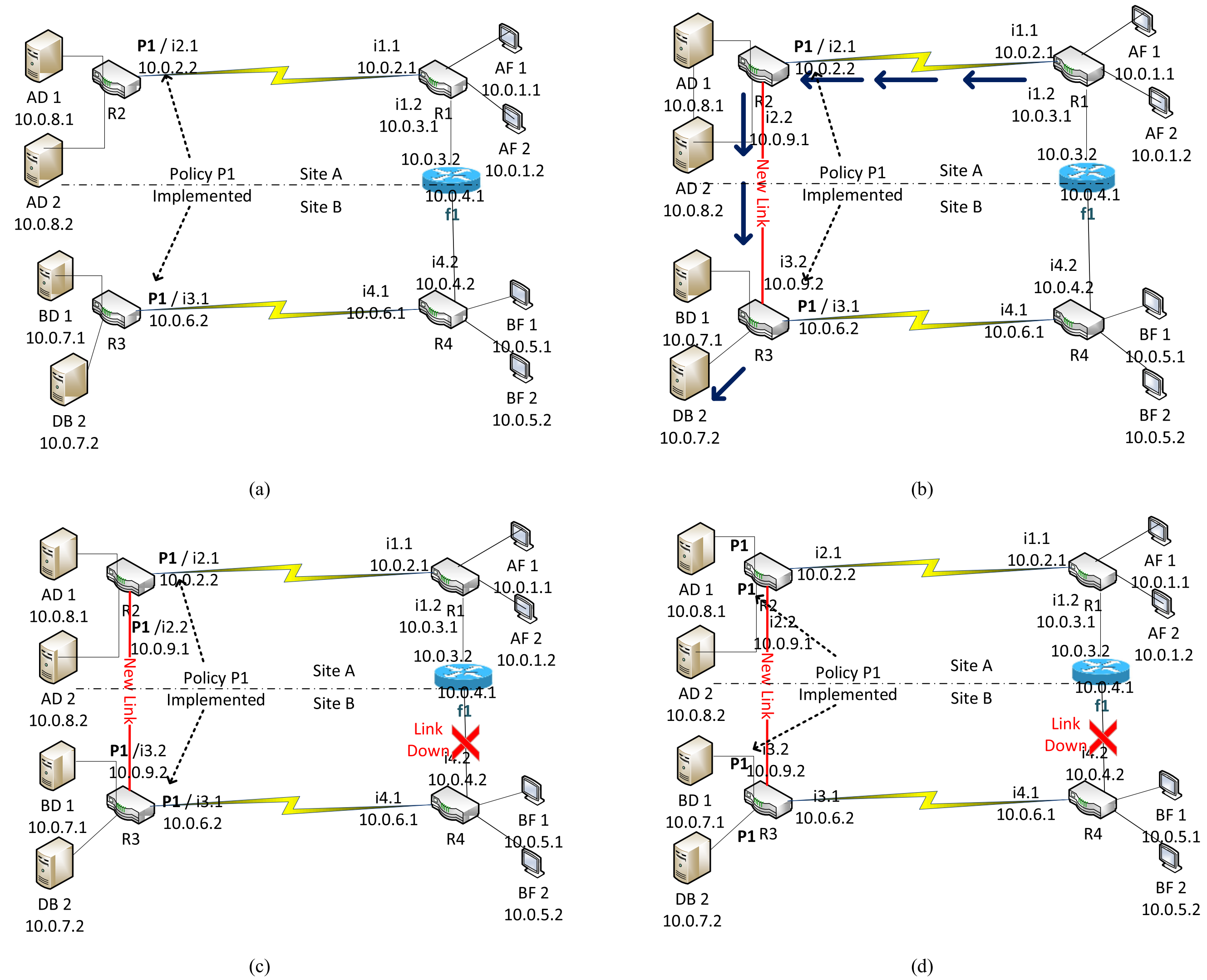


FIGURE 1. (a) A network for Case 1. (b) A network for Case 2. (c) A network for Case 3. (d) A network for improved policy implementation.

cases(Case 1, Case 2 and Case 3). Thus, there is a need of a customized approach to be implemented at the SDN controller in order to avoid ACL policy violation in the case of a topology change in hybrid SDN.

III. RELATED WORK

Veriflow [4] presents the technique for checking in real time the network invariants like loops, black holes, and access control verification. After detecting these network invariant, one can raise alarm or can block these events. It is implemented for pure SDN architecture and resides between SDN controller and forwarding devices as indicated in Figure 2. Veriflow has three steps (i) by using network policy rules, the network is divided into slices by generating a set of equivalence class (EC). An EC represents the set of packets that experience the same forwarding actions through the network. (ii) Veriflow builds individual forwarding graphs for every modified EC that represent network modified behavior. (iii) Veriflow uses these graphs to determine the status of one or more invariants. Veriflow is made for pure SDN and thus it cannot be used in hybrid SDN. Because Hybrid SDN has switches which require a customized mechanism to operate them. Veriflow has difficulty in verifying network invariants in real-time when large numbers of ECs are altered in one operation, and when there is link failure.

Header Space Analysis (HSA) [13] presents the technique that helps the system administrators to statically analyze the SDN. HSA assumes that a network can be divided into different groups. Each group has a set of hosts, users and traffic class that are isolated from other groups. For example, ‘‘Can I prevent host A from talking to host B’’ so a network application should slice the network into two groups and ensures that packets originated from one group should not reach to another group. HSA provides a set of methods and operations that examines various failure conditions (like reachability, loop detection) regardless of a specific communication protocol

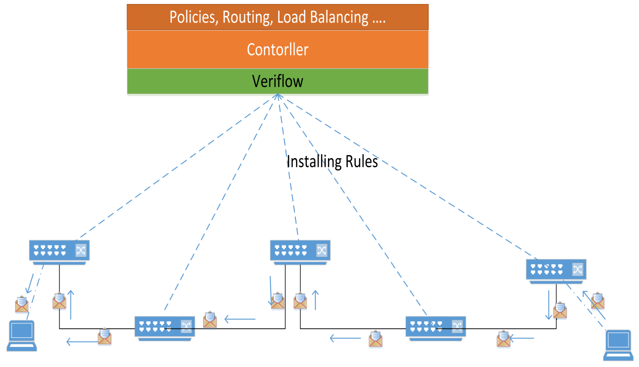


FIGURE 2. Veriflow.

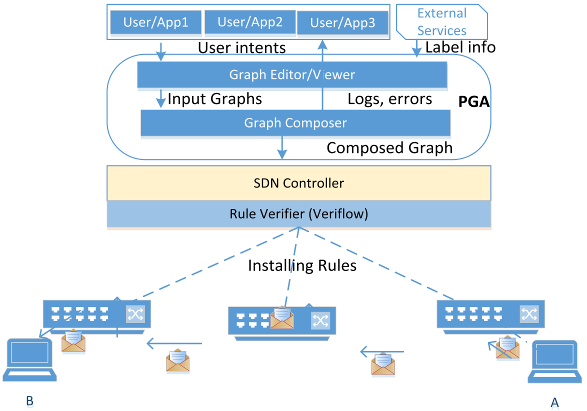


FIGURE 3. PGA system architecture.

implemented in the network. HSA adopts a general geometric approach for packets classification. Packets are rendered as points in a geometric space and switches are considered as transfer functions on the space. In this way, a packet forwarding in the network can be seen as a point moving in the geometric space from one place to another place. HSA further defines several types of operations on the point in the space and initiates the optimization for computation. HSA can also detect different network invariants such as node reachability or loop detection that can be regarded as tracking traces of points in the geometric space. HSA does not deal with access control violation in the case of a topology change in hybrid SDN. Moreover, this mechanism is limited for pure software defined network architecture and not feasible for hybrid SDN.

HybNET [6] presents a network management framework for hybrid SDN. It provides central control and management over legacy and Openflow based switches through virtualization across the whole network. It tries to hide the dissonance between legacy and SDN network configurations by providing a common interface to the central controller. For this purpose, virtual links are composed of multiple links between Openflow and legacy devices. In HybNET, Openflow switches provide the main tasks of network management and control with the help of a controller. Legacy switches are working as forwarding devices only. This provides the network virtualization functionality using VLAN in the traditional network and using fine-grained forwarding rules installed by the SDN controller. HybNET is implemented in OpenStack [15] which is the most popular open source cloud computing platform by using neutron [16] **as** the network service of OpenStack for host side network virtualization. The physical hybrid network is formed in HybNET by using Openflow and legacy switches. HybNET focus as neither on topology change nor access control violation problem.

PGA [5] is a technique for automatic composition of high-level network policies (e.g. load balancing, ACL, etc.). It examines multiple individually specified network policies for any conflict among these policies. In a large organization, there are a number of policy subdomains like server administrator, network engineer, DNS domains, etc. To change a network policy, it takes a long time, like days to weeks. Because this requires extreme care to implement the policy in such a way that new policy does not have a conflict with an existing policy. For example, a company wants to implement a CRM (Customer Relation Management) [17] application for customers in its office. According to policy P1 only marketing employees can send data to CRM servers using port 7000 and through a load balancing service (LB). Another network wide policy P2 defines that employees have restricted access to company servers through TCP port 80, 334 and 7000 and traffic should pass through the firewall. These two policies P1 and P2 need to be combined into a consistent single policy that maintains perspective of both policies. In some automated network infrastructures, i.e. enterprise networks, clouds and Network Functions Virtualization (NFV) [14], policies are generated automatically and the policies are many in number. To detect the conflict among network policy, PGA uses a graph based abstraction to simplify the network policies. In PGA, users, administrator, and SDN applications define their network policies and these policies are specified in the form of graphs. These graphs are submitted to Graph Composer through a PGA User Interface (UI). Graph composer automatically resolves the conflict among different graphs. The graph composer can give some possible suggestions to the network administrator to resolve the conflicts among various policies and finally generates an error-free/conflict free graph. The overall system of PGA is shown in Figure 3. This paper just focuses on the conflict in the implementation of multiple policies and resolves these conflicts. It does not deal with policy configuration.

Telekinesis [18] is a network controller that does efficient route management activities in the hybrid SDN. Telekinesis introduced a new flow control, called LegacyFlowMod, to control the routing activities in both legacy and Openflow switches. Telekinesis instructs the Openflow switches to send special packets to legacy switches to update the forwarding entries at legacy switches in such a way that as soon as a data packet enters the network is forwarded to nearest Openflow switch. This is an attempt to forward data packet to the controller as soon as possible so that the controller process

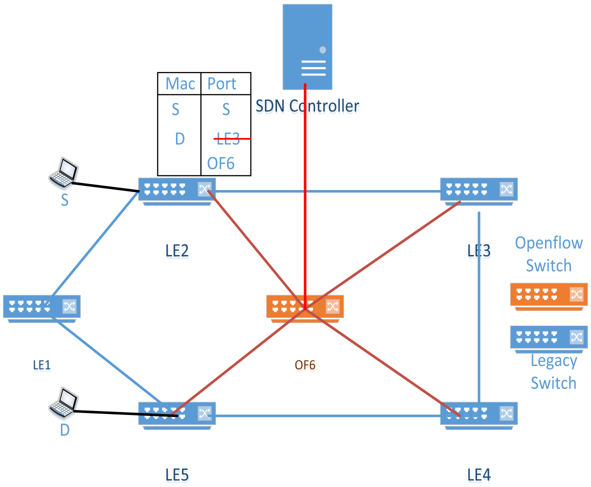


FIGURE 4. Overview of telekinesis.

the packet and subsequently installs flow entries in the concerned switches. For example, a path from LE2 to LE3 must pass from an Openflow switch OF3. In this way, by using Openflow and Mac Learning, packets received on a specific port of legacy switches are forwarded to the respective SDN switches as shown in figure 4.

Telekinesis focuses only on the routing control in hybrid SDN and does not consider ACL violation in the case of topology changes.

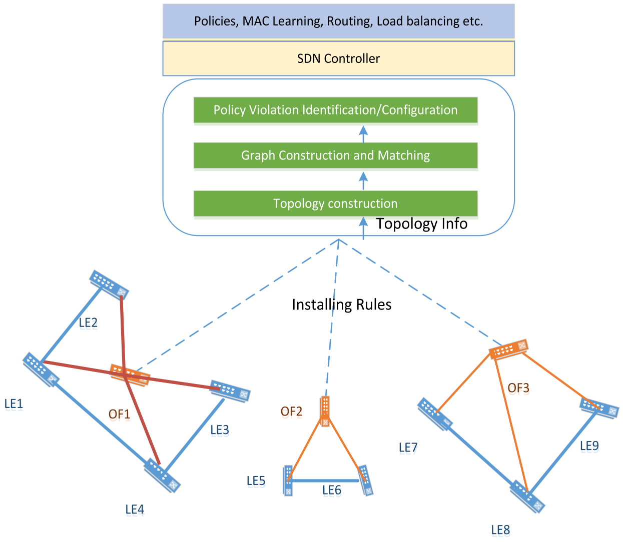
Exodus [19] presents control system for legacy network devices that gets device configuration and compile them into an intermediate form and finally, produces the equivalence SDN rules. Exodus provides a solution for traditional networking to migrate on SDN by translating traditional networking configuration (e.g. Cisco IOS) to Openflow based configuration. Exodus takes the IOS configuration (maybe from different routers) as input and passed to Exodus IOS Parser and Compiler that generates the network Specification and Flowlog libraries that can be used as a prototype for Openflow rules. In this way, traditional networking devices can be used to perform as Openflow like infrastructure. This paper deals with configuration translation from legacy devices to Openflow control information. However, Exodus does not consider ACL violation in the case of a change in network topology.

From the above literature, it is clear that ACL violation in the case of a change in network topology has not been discussed in hybrid SDN. It is a big issue because, in a communication network, topology changes frequently that affects both the network control and performance [20], [27]. Due to a topology change, packets may be traveled to an unauthorized node if the security policy is not verified for the updated topology. Thus, security of network is at risk as shown through some examples in Section II. To auto-configure ACL policies in hybrid SDN, there are following challenges.

- Auto-identification of proper switches and interfaces where policies are to be implemented
- In case of topology change, auto-detection of topology change
- Auto-identification of policy violation
- Pointing out the interfaces where policy violate in case of topology change
- Which policy is violated in case of topology change?
- Reconfiguring policy as per specification

IV. PROPOSED SOLUTION

In order to handle the problems as discussed in Section II, we proposed an automatic network policy enforcement mechanism called Auto-PDTC, which detects policy violation in the case of network topology changes and enforces the policy in an optimum way by configuring the policy on the affected device's interfaces. We model the Hybrid SDN (CN) as CN = (E, N), where E is a set of the undirected edges and N denotes the set of nodes. N is subdivided into two subsets; L consists of legacy switches, and F consists of both Openflow based switches and controller. Thus, $N = L \cup F$. A path from source $s \in N$ to destination point $t \in N$ such that $s \neq t$ is represented as a list of traversed links, the mathematical path is represented as $p(s, t) = \{s, v1, v2 \ldots vk, t\}$ and where $v1, v2 \ldots vk\ L \cup F$.



Policies, MAC Learning, Routing, Load balancing etc.
SDN Controller
Policy Violation Identification/Configuration
Graph Construction and Matching
Topology construction
Topology Info
Installing Rules
LE2
LE1
OF1
LE3
LE4
OF2
LE5
LE6
OF3
LE7
LE9
LE8

FIGURE 5. Overall system design

In Figure 5, we have shown the overall system design components and their interaction with the external environment. As we have discussed earlier, Auto-PDTC policy enforcement system consists of three main components. The first component is topology construction. In this component, network topology of devices and links is constructed. This topology information is passed to next component. The second component is graph construction and graph matching. This fetches the topology information from the first component, generates the graph using graph construction algorithm and also detects the difference between the graphs that are constructed at different time intervals. The third component is policy verification that is used to detect policy violation due to the change in network topology and subsequently configures the policies on affect interfaces.

Our prototype support hybrid SDN environment in which there are a number of legacy switches and among them, a limited number of Openflow switches are placed. We used POX Openflow controller to generate the Openflow rules for packet-in event based on network policies implemented by the network administrator.

We model network-wide policy and local policy at forwarding device using a 3-tuple and a 6-tuple, respectively. We compute graph to represent the topology of network. By using Graph Difference [11], we detect a possible change in topology. In the case of topology change, we verify the policy for updated topology by traversing the tree using 6-tuple. If there is any violation in policy implementation, then affected interfaces are indicated and policies that need to be configured are also indicated. Then policies are configured on the updated topology according to specification in an improved way. The detail of each component of our proposed solution is explained as follows.

A. NETWORK POLICY REPRESENTATION

Through a 3-tuple <source IP, Access Policy, Destination IP>, we represent a network wide policy in our system. In this tuple, Source IP represents the policy for a packet originated from Source IP, Access Policy describes the specific policy to drop (not allowed) or forward (allowed) the packet and Destination IP means the machines to which the packet is not allowed or allowed. For example, 3- tuple <10.0.1.1 – 10.0.1.2, Not Allowed, 10.0.7.1-10.0.7.2> means that packets originated from IP address 10.0.1.1 - 10.0.1.2 are not allowed to access devices with IP 10.0.7.1- 10.0.7.2, for Case 1. The network-wide policy is translated into local policies which are installed at switches/router's interfaces. The local policy is represented in our system by 6-tuple <Router/Switch, Source IP Address, Port, Policy, Allowed Port, Not Allowed Ports> used as follows. For policy, if a packet Pkt originated from Source IP Address is received at Router/Switch say R1 at R1's physical Port, then Pkt can be forwarded as per Policy to the Allowed Ports of R1 and Pkt cannot be forwarded (i.e. blocked) to Not Allowed Ports. For example, for Case 1, 6-tuple <R3, 10.0.1.1-10.0.1.2, i3.1, P1, 2-4, 1> means that at router R3 if Pkt originated from 10.0.1.1-10.0.1.2 is received at interface i3.1 then Pkt is not allowed to be forwarded to ports i3.2-i3.3 and not allowed to port i3.1.

B. TOPOLOGY CONSTRUCTION

We get the network topology of Openflow and legacy devices at the controller as follows. An Openflow device (switch/router) exchanges periodically its link state information with a controller. The controller gets remote log information of legacy devices (switches/routers) to get their link state information. Thus, after getting the link state information from all forwarding devices, the edges are stored in a set E and the nodes are stored in a set V. We construct an undirected graph G where forwarding devices are represented as nodes, and links are represented as edges.

Algorithm 1 Graph Construction

```
Input: E is number of edges, V is number of vertices
Output: Graph G
G= {0} // G is empty
while (the Instance is not solved)
    Select the edge from the E and vertices from
    if edge connects two vertices in disjoint subsets then
        merge the subsets;
        add the edge to G;
    end if
    if all the subsets are merged then
        the instance is solved
    end if
end while
```

In graph construction algorithm, an edge from E and its respective vertices are selected and added to graph G. Then next edge and its respective vertices are selected, and then added to G. This process is repeated till all edges and vertices are added to G. Algorithm 1 explains the graph construction.

C. DETECTING CHANGE IN NETWORK TOPOLOGY

If network topology is changed as shown in Figure 6(b), then we detect this change in topology with the help of graph difference algorithm [11], [28]. Let G1= (V1, E1) represents communication network topology over a time t1. In this tuple, V1 denote the vertices and E1 denote the edges. Let at another time instance t2 graph G2 is computed for the topology. If G2 is isomorphic to G1, then edge structure is preserved in both the graphs and thus there is no change in the topology of the network. Otherwise, there is a change in the topology. For Figure 6(a) and Figure 6(b), we can detect the difference of edges using mapping of edges of both figures.

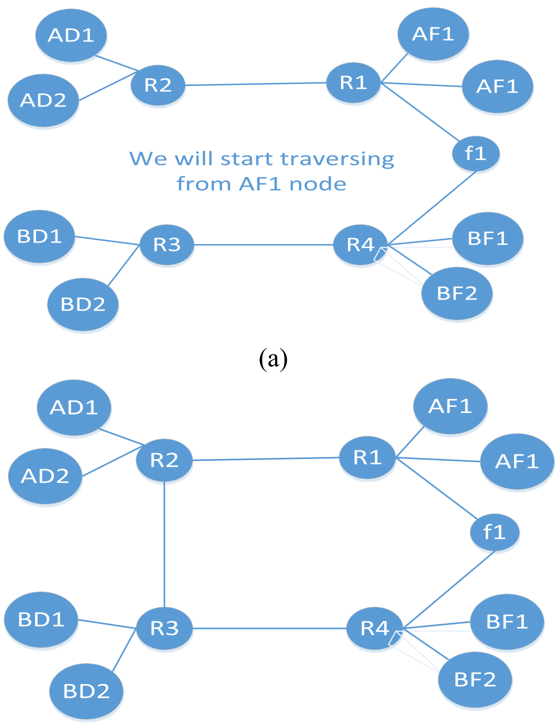
Graph difference algorithm works as follows.

Definition 1: A triple $G = (V, E, \mu)$ represent an undirected and labeled graph G for communication network topology over a time t_1 where:

- V is a finite set of vertices
- $E \subseteq V \times V$ is set of edges and $e(i, j)$ represents a directed edge transmitting traffic from vertex i to j.
- $\mu : V \rightarrow L_v$ is a function assigning unique labels to each vertex in G such that $\mu(i) \neq \mu(j)$

Graph isomorphism exists between two graphs if both graphs are same. Two graphs which contain the same number of graph vertices connected in the same way are said to be isomorphic. If G_1 and G_2 are two graphs, then graph isomorphism can be detected by mapping vertices of one graph G_1 onto vertices of second graph G_2.

Definition 2: A bi-jective function f: $V_1 \rightarrow V_2$ is a graph isomorphism from $G_1 = (V_1, E_1, \mu_1)$ and $G_2 = (V_2, E_2, \mu_2)$, if



(a)

(b)

FIGURE 6. **(a) Graph for Case 1. (b) Graph for Case 2.**

- $\mu_1(\mathrm{v}_1) = \mu_2(f(\mathrm{v}_1))$, $\mathrm{v}_1 \epsilon \mathrm{V}_1$
- For all v_1, $\mathrm{v}_2 \epsilon \mathrm{V}_1$, the edge $\mathrm{e}_1 = (\mathrm{v}_1, \mathrm{v}_2) \epsilon \mathrm{E}$ if and only if the edge $\mathrm{e}_2 = (\mathrm{f}(\mathrm{v}_1), \mathrm{f}(\mathrm{v}_2)) \epsilon \mathrm{E}_2$

There are a number of techniques [21]–[23] to measure the degree of similarity between two graphs. We used error correcting graph matching (ECGM) technique to detect the graph isomorphism. ECGM measures the difference between two graphs by computing the minimal sequence of edit operations required to convert one graph G_1 into another graph G_2. Edit operations include insertion, deletion, and substitution of edges or vertices. These edit operations are calculated while converting one graph G_1 to another graph G_2 as a similarity measure.

To derive an expression for similarity measure based on ECGM, the following definitions are used.

Definition 3: Given a graph G = (V, E, μ), define all possible edit operations δ on G as follows:

- ($\mathrm{v} \rightarrow \$$), v ϵ V: deleting a vertex v from G and all edges in G that are incident to v
- ($\$ \rightarrow \mathrm{v}$): inserting a vertex v into G with the unique vertex label μ (v) ϵ $\mathrm{L}_\mathrm{v}(\mathrm{v})$
- ($\mathrm{e} \rightarrow \$$), e ϵ E: deleting the edge from G
- ($\$ \rightarrow \mathrm{e}$), $\mathrm{e} = (\mathrm{v}_1, \mathrm{v}_2)$ and v_1 and $\mathrm{v}_2 \epsilon$ V: inserting an edge between two vertices v_1 and v_2

Definition 4: Given a graph G = (V, E, μ) and an edit operation δ on G, the edited graph δ(G) becomes the graph $\delta(\mathrm{G}) = (\mathrm{V}_\delta, \mathrm{E}_\delta, \mu_\delta)$ where:

$$V_\delta = \begin{cases} V - \{v\} & if\ \delta = (\mathrm{v} \rightarrow \$) \\ VU\,\{v\} & if\ \delta = (\$ \rightarrow \mathrm{v}) \\ V & otherwise \end{cases}$$

$$E_\delta = \begin{cases} E - \{e\} & if\ \delta = (\mathrm{e} \rightarrow \$) \\ EU\,\{e\} & if\ \delta = (\$ \rightarrow \mathrm{e}) \\ E & otherwise \end{cases}$$

$$\mu_\delta = \begin{cases} \mu|V - \{v\} & if\ \delta = (\mathrm{v} \rightarrow \$) \\ ext.of\,\mu \text{ to V } U\,\{v\} & if\ \delta = (\$ \rightarrow \mathrm{v}) \\ \mu & \text{otherwise} \end{cases}$$

Definition 5: Given a graph G= (V, E, μ) and a sequence of edit operations$\Delta = (\delta_1, \delta_2, \ldots.., \delta_k)$, k $\geq$ 1, the edited graph Δ (G) becomes

$$\mathbf{\Delta(G) = \delta_k(\ldots.\delta_2(\delta_1(G))\ldots)}$$

If we assign a cost $\mathrm{W}(\delta_\mathrm{i})$ for every edit operation, then total cost related to this sequence of edit operation Δ is

$$\mathrm{W}(\Delta) = \sum\nolimits_{i=1}^{k} W(\delta \mathrm{i})$$

Definition 6: Given two graphs $\mathrm{G}_1 = (\mathrm{V}_1, \mathrm{E}_1, \mu_1)$ and $\mathrm{G}_2 = (\mathrm{V}_2, \mathrm{E}_2, \mu_2)$, and Δ represents the series of edit operations on G_1 such that Δ (G_1) is graph isomorphic to G_2, the edit distance $d(G_1, G_2)$ between two graphs is the minimum sum of costs associated with edit sequence Δ

$$d(G_1, G_2) = W(\Delta)$$

Definition 7: Suppose the graph $\mathrm{G}_1 = (\mathrm{V}_1, \mathrm{E}_1, \mu_1)$ represent the communication network functioning at time t_1 and $\mathrm{G}_2 = (\mathrm{V}_2, \mathrm{E}_2, \mu_2)$ describe the same network as time t_2 where $\mathrm{t}_2 = \mathrm{t}_1 + \Delta$ t. The network *edit distance* $d(G_1, G_2)$ can be defined as :

$$d(G_1, G_2) = |\mathrm{V}_1| + |\mathrm{V}_2| - 2|\mathrm{V}_1 \cap \mathrm{V}_2| + |\mathrm{E}_1| \\ + |\mathrm{E}_2| - 2|\mathrm{E}_1 \cap \mathrm{E}_2|$$

Where the cost function for edit operations δ is

$$W(\delta) = \begin{cases} 1 & \delta = (\mathrm{v} \rightarrow \$) \\ 1 & \delta = (\$ \rightarrow \mathrm{v}) \\ 1 & \delta = (\mathrm{e} \rightarrow \$) \\ 1 & \delta = (\$ \rightarrow \mathrm{e}) \\ 0 & otherwise \end{cases}$$

We can see here that the edit distance is used as a measure of change in topology of the network and edit distance increases in case of more change in network topology over time Δ t. Edit distance $d(G_1, G_2)$ of two graphs is bounded and $d(G_1, G_2) = 0$ when G_1 *and* G_2 *are isomorphic* means that there is no change in topology. The algorithm to find the change in two graphs G_1 (the network topology at t_1) and $_\mathrm{G}$ (the network topology at t_2) is given in Algorithm 2.

If there is a change in the topologies (i.e. G_1 and G_2), then we will traverse the updated network topology

Algorithm 2 Graph Difference Algorithm

```
Input: Non-empty attributed graphs G1 = (V1, E1, $\mu$1) and G2 = (V2, E2, $\mu$2) where V1 = {u1, ..., un} and V2 = {u2, ..., u|v2|}
Output: A minimum cost edit path (pm) from G1 to G2 e.g., {u1 $\rightarrow$v3, u2 $\rightarrow$ $\varepsilon$, $\varepsilon$$\rightarrow$v2}
New $\leftarrow \{\varphi\}$, $P_m \leftarrow \varphi$
For each node $w \in V_2$, New $\leftarrow$ New $\cup \{u_1 \rightarrow w\}$
    New $\leftarrow$ New $\cup \{u_1 \rightarrow \varepsilon\}$
end For
while (true) do
    $P_m \leftarrow \arg_m\{g(p) + lb(p)\}$ s.t. $p \in$ New
    New $\leftarrow$ New $\setminus P_m$
    if $P_m$ is a complete edit path then
        return $P_m$ as a solution (i.e., the minimum cost edit
        distance from $G_1$ to $G_2$)
        else
        Let $P_m \leftarrow \{u_1 \rightarrow v_{i1}, \ldots, u_k \rightarrow v_{ik}\}$
        if $k < |V_1|$ then
            For each $w \in V_2 \setminus \{v_{i1}, \ldots, v_{ik}\}$, New $\leftarrow$ New $\cup \{P_m \cup \{u_{k+1} \rightarrow w\}\}$
                $P_{new} \leftarrow P_m \cup \{u_{k+1} \rightarrow \varepsilon\}$
                New $\leftarrow$ New $\cup \{P_{new}\}$
            end For
            else
            $P_{new} \leftarrow P_m \cup U_{w \in V2 \setminus \{v_{i1}, \ldots, v_{ik}\}} \{\varepsilon \rightarrow w\}$
            New $\leftarrow$ New $\cup \{P_{new}\}$
        end if
    end if
end while
```

(i.e. the graph G2) for the policy violation. For this purpose, we propose to construct a search tree by using both network topology G_2 and 6-tuple. Policies are implemented on the interfaces of forwarding devices (routers/switches) by starting from an end system (i.e. AF1, AF2, BF1, BF2, AD1, AD2, BD1, and BD2). That is, we model the network topology G_2 and 6-tuple policies in the tree form by defining the interfaces of a forwarding device as branches of the tree.

The branch of the tree is defined as *allowed*if the traffic is allowed/can be forwarded on these interfaces. Otherwise, the branch is defined as *not allowed* as per 6-tuple policy. This indicates the blocking and non-blocking paths for a packet at the forwarding device. For example, for Case 1, we want to explore that, packet originated from AF1-AF2 can reach to which part of the network as per 6-tuple <Router/Switch, Source IP Address, Port, Policy, Allowed port, Not Allowed Ports> policy P1 implemented on the interfaces of forwarding devices. In this scenario, a packet Pkt originated by AF1 will reach to R1 router and Pkt can be forwarded to all other ports of router R1 as indicated by 6-tuple at R1, this is shown in Figure 7(a) model according to the proposed approach. Then Pkt reaches to the router f1 which is Openflow router. After this, f1 forwards to router R4 as shown in Figure 7(b). R4 can forward to three ports (i.e. allowed ports are 10.0.5.1, 10.0.5.2 and 10.0.6.1) as shown in Figure 7(c). Similarly, after reaching at R3 at interface i3.1, the Pkt is not allowed to be forwarded as shown in Figure 7(c), i.e. thus Pkt will be dropped here. Similarly, for Case 2, when the topology is updated due to the addition of new link from R2 to R3. Then we traverse new topology and find policy violation for the AF1-AF2 subnet on interface i3.2 as shown in Figure 7(d). In Case 3, when there is a link breakage between f1 and R4 then AF1-AF2 subnet cannot communicate with BF1-BF2 subnet although there is path present in through R1, R2, R3 and R4, as shown in Figure 7(e). Thus, in this case, the optimum way of policy implementation will be to implement P1 at R3's interfaces BD1 and BD2, as shown in Figure 1(d) and its policy tree is shown in Figure 7(f). The algorithm for modeling both network topology and 6-tuple policy in the tree form is given in Algorithm 3.

Algorithm 3 Algorithm for Modeling Both Network Topology and 6-Tuple Policy in the Tree Form

```
Input: $G_2 = (V_2, E_2)$ where $V_2 = \{v_1, v2 \ldots v_k\}$ are vertices and $E_2 = \{e_1, e_2 \ldots\ldots e_k\}$ are the edge of the updated network topology, 6-tuple policy P1, Pkt originated from Host AF1, AF2, BF1, BF2 etc.
Output: Policy violation either exists or not.
Pkt.src
For $h_1$ to $h_n$
    model both $G_2$ and $P_1$ for Pkt originated from $h_i$
    if (Policy violation exists) then
        Point Out (Generate Alarm)
        else
        ][ Select appropriate path for packets
    end if
end For
```

In Figure 8, we have shown the flowchart for overall system operation. First, we get link state information from all switches and nodes, and then the graph is constructed for topology. After this graph difference is calculated. if there is a change in the graph then we use tree traversing to detect the affected interfaces for policy violation. In the case of policy violation, proper policy configuration is performed.

Our proposed solution also addresses other network invariants like reachability and loop detection. For example, in the scenario shown in Figure 9, when node AF1 sends six packets to node BF1. Suppose after first two packets passed through f1, the outgoing link goes down. Then remaining packets are dropped in dropped at f1 in the existing approaches. However, by using our proposed solution, we can traverse the

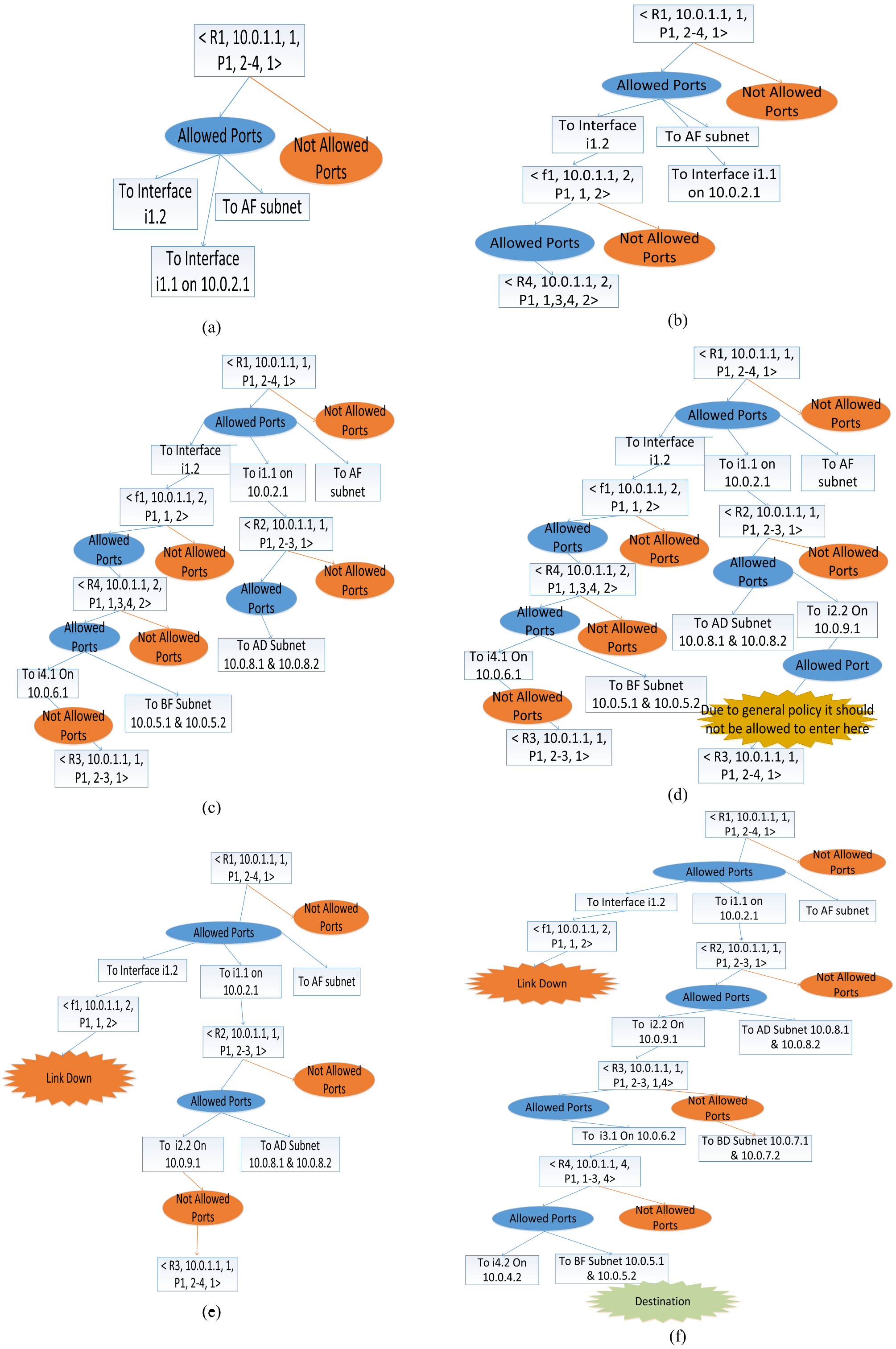

< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To AF subnet
To Interface i1.1 on 10.0.2.1
(a)
< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To AF subnet
To Interface i1.1 on 10.0.2.1
< f1, 10.0.1.1, 2, P1, 1, 2>
Allowed Ports
Not Allowed Ports
< R4, 10.0.1.1, 2, P1, 1,3,4, 2>
(b)
< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To i1.1 on 10.0.2.1
To AF subnet
< f1, 10.0.1.1, 2, P1, 1, 2>
< R2, 10.0.1.1, 1, P1, 2-3, 1>
Allowed Ports
Not Allowed Ports
< R4, 10.0.1.1, 2, P1, 1,3,4, 2>
Allowed Ports
Not Allowed Ports
To AD Subnet 10.0.8.1 & 10.0.8.2
To i4.1 On 10.0.6.1
To BF Subnet 10.0.5.1 & 10.0.5.2
Not Allowed Ports
< R3, 10.0.1.1, 1, P1, 2-3, 1>
(c)
< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To i1.1 on 10.0.2.1
To AF subnet
< f1, 10.0.1.1, 2, P1, 1, 2>
< R2, 10.0.1.1, 1, P1, 2-3, 1>
Allowed Ports
Not Allowed Ports
< R4, 10.0.1.1, 2, P1, 1,3,4, 2>
Allowed Ports
Not Allowed Ports
To AD Subnet 10.0.8.1 & 10.0.8.2
To i2.2 On 10.0.9.1
To i4.1 On 10.0.6.1
To BF Subnet 10.0.5.1 & 10.0.5.2
Allowed Port
Due to general policy it should not be allowed to enter here
Not Allowed Ports
< R3, 10.0.1.1, 1, P1, 2-3, 1>
< R3, 10.0.1.1, 1, P1, 2-4, 1>
(d)
< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To i1.1 on 10.0.2.1
To AF subnet
< f1, 10.0.1.1, 2, P1, 1, 2>
< R2, 10.0.1.1, 1, P1, 2-3, 1>
Link Down
Allowed Ports
Not Allowed Ports
To i2.2 On 10.0.9.1
To AD Subnet 10.0.8.1 & 10.0.8.2
Not Allowed Ports
< R3, 10.0.1.1, 1, P1, 2-4, 1>
(e)
< R1, 10.0.1.1, 1, P1, 2-4, 1>
Allowed Ports
Not Allowed Ports
To Interface i1.2
To i1.1 on 10.0.2.1
To AF subnet
< f1, 10.0.1.1, 2, P1, 1, 2>
< R2, 10.0.1.1, 1, P1, 2-3, 1>
Link Down
Allowed Ports
Not Allowed Ports
To i2.2 On 10.0.9.1
To AD Subnet 10.0.8.1 & 10.0.8.2
< R3, 10.0.1.1, 1, P1, 2-3, 1,4>
Allowed Ports
Not Allowed Ports
To i3.1 On 10.0.6.2
To BD Subnet 10.0.7.1 & 10.0.7.2
< R4, 10.0.1.1, 4, P1, 1-3, 4>
Allowed Ports
Not Allowed Ports
To i4.2 On 10.0.4.2
To BF Subnet 10.0.5.1 & 10.0.5.2
Destination
(f)

FIGURE 7. Tree computation. (a) Tree computation at R1 for a packet originated from AF1-AF2 for case1. (b) Tree computation at f1 after traversing from R1, for a packet originated from AF1-AF2 for case 1. (c) Tree computation at R4 after traversing from R1 and f1 for a Packet originated from AF1-AF2 for case 1. (d) Complete Tree computed for a packet originated from AF1-AF2 for case 2. (e) Complete Tree computed for a packet originated from AF1-AF2 for case 3. (f) Complete Tree computed for a packet originated from AF1-AF2 for Figure 1(c).

tree from AF1, find the alternative route f1, R1, R2 and R3, then we forward remaining packets through this alternative route.

Similarly, loops can also be detected by using our tree computation method. In this method, we can traverse the whole network for possible looping condition and by

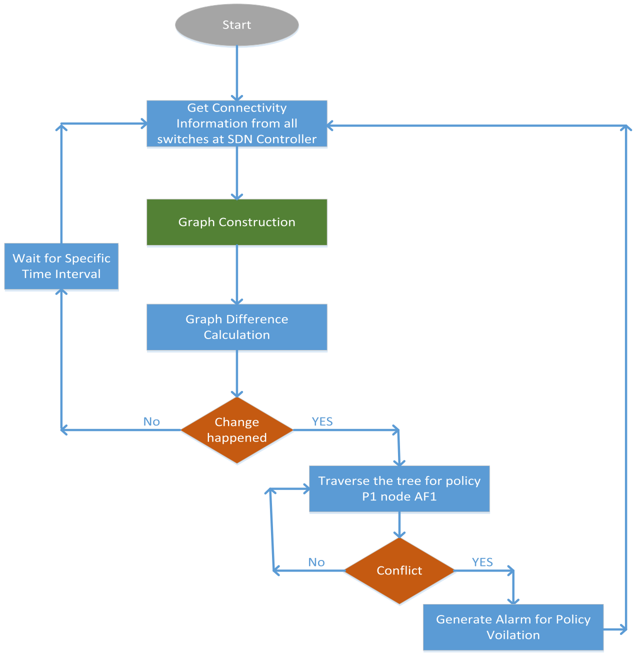

Start
Get Connectivity Information from all switches at SDN Controller
Graph Construction
Wait for Specific Time Interval
Graph Difference Calculation
No
Change happened
YES
Traverse the tree for policy P1 node AF1
No
Conflict
YES
Generate Alarm for Policy Voilation

FIGURE 8. Flow chart for different cases of topology change.

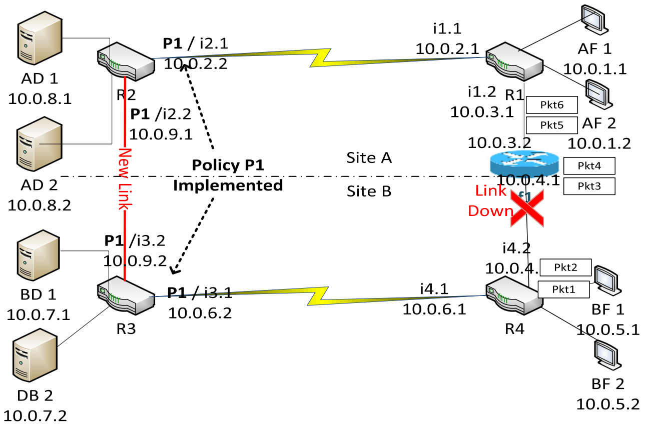


FIGURE 9. Reachability scenario.

implementing a specific rule on interfaces we can avoid the loops.

V. PERFORMANCE EVALUATION

For performance evaluation, we used Mininet [10] with POX controller [24]. All of our experiments are performed on an HP Envy 14-j102tx machine with an Intel Core i5 6200 CPU with 4 physical cores and 8 threads at 2.4 GHz speed, and 8 GB of RAM, running 64 bit Ubuntu Linux 16.01 LTS. We used Openflow switches and legacy switches to construct a topology of the system [29]. To use OpenvSwitch (OVS) as a legacy switch, we set the ovs fail mode to be "standalone" and disconnected it with the controller. We study the scenarios of 1, 2, 3, 4 and 5 number of links failure or update, in a random way, for change in network topology. We evaluate the performance of our proposed approach using following parameters by varying frequency of links failure/addition, number of switches, data rate and frequency of getting network topology information at the controller.

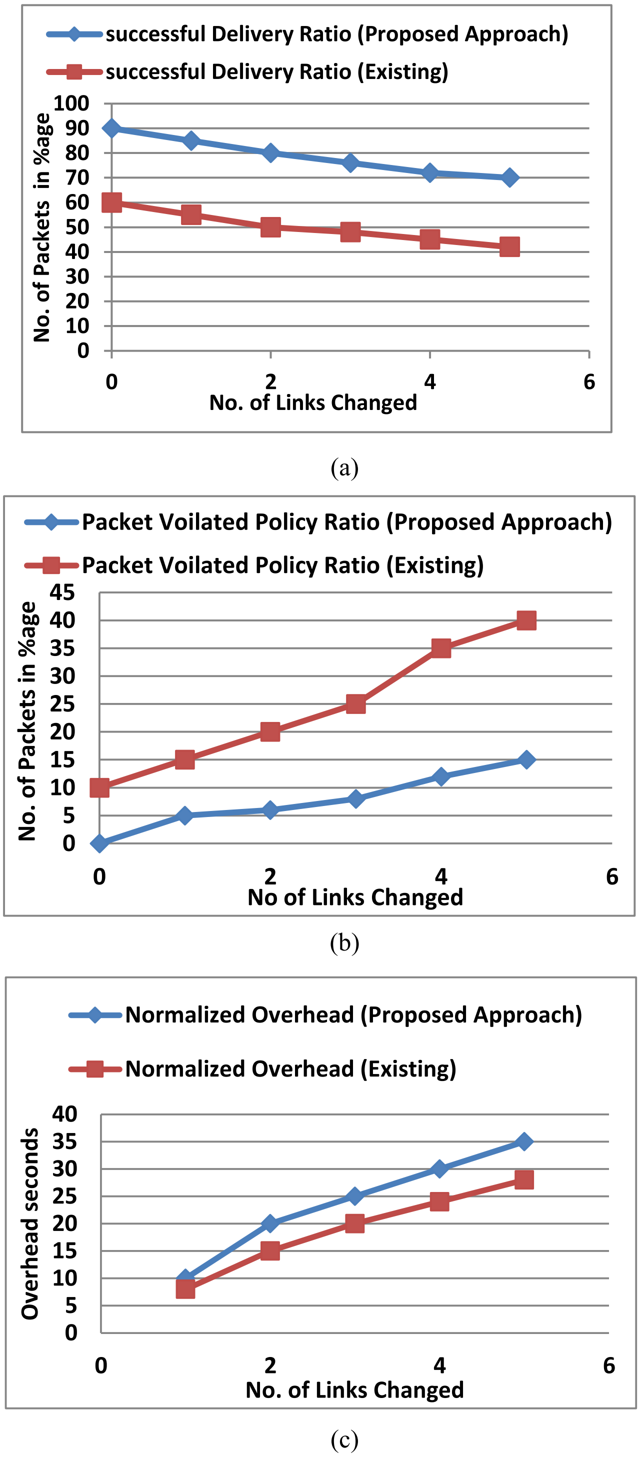
- Successful Packet Delivery (SPD) ratio: The total number of packets received at destinations as per policy to the total number of packets initiated at source nodes.
- Packets Policy Violated (PPV) ratio: Total number of packets that violate the policy to the total number of packets initiated at source nodes.
- Normalized overhead: Total number of transmissions in the network divided by total number of packets received successfully at destination nodes as per policy.

We assumed that nodes try to violate the policy by sending packets randomly to different destination nodes or by broadcasting the packet. The approach that is adopted in [1], we call it as existing approach. In this technique, network topology change is detected by the system administrator and proper identification of affected interfaces is performed. It is necessary for a network administrator to implement policies on new interfaces according to specification. For this purpose, firstly, the network administrator will translate the network policy to corresponding network rules. Secondly, this policy will be configured on switches using ACL commands [30]. This task requires a lot of time to implement policies properly on respective interfaces.

A. RESULTS BY VARYING FREQUENCY OF LINK FAILURE/ADDITION

Failure and addition of links are the most common events that occur very frequently in a communication network. Figure 10(a) shows that SPD ratio in our proposed approach is higher than existing one because our proposed approach automatically detects the change in topology and implements policies on affected interfaces at an early stage. Thus, our proposed approach avoids the occurrence of Case 3. By enforcing the policies at an early stage at the interfaces of forwarding devices, our approach stops the packets to violate the policy. The SPD ratio gets decreased in both approaches (i.e. in our approach and the existing one) as the frequency of link failure/addition in the network increases. This is due to following reasons. (i)By adding more number of new links, the probability that the policy is violated by a packet gets increased, as discussed in Case 1. Subsequently, this results in the violation of policies by more number of packets. This reduces the SPD ratio of both approaches. (ii)When a number of links get failed in the network, the chances of Case 2 increases which in turn reduces the SPD.

The number of packets that violate network policy due to change in topology are also computed and the corresponding results are shown in Figure 10(b). Figure 10(b) indicates that our proposed approach has a lower ratio of packets that violated the policy during topology change and implementation of policy on new interfaces. We also examine the case when some packets are dropped during the detection of policy violation and policy implementation on the new interfaces in

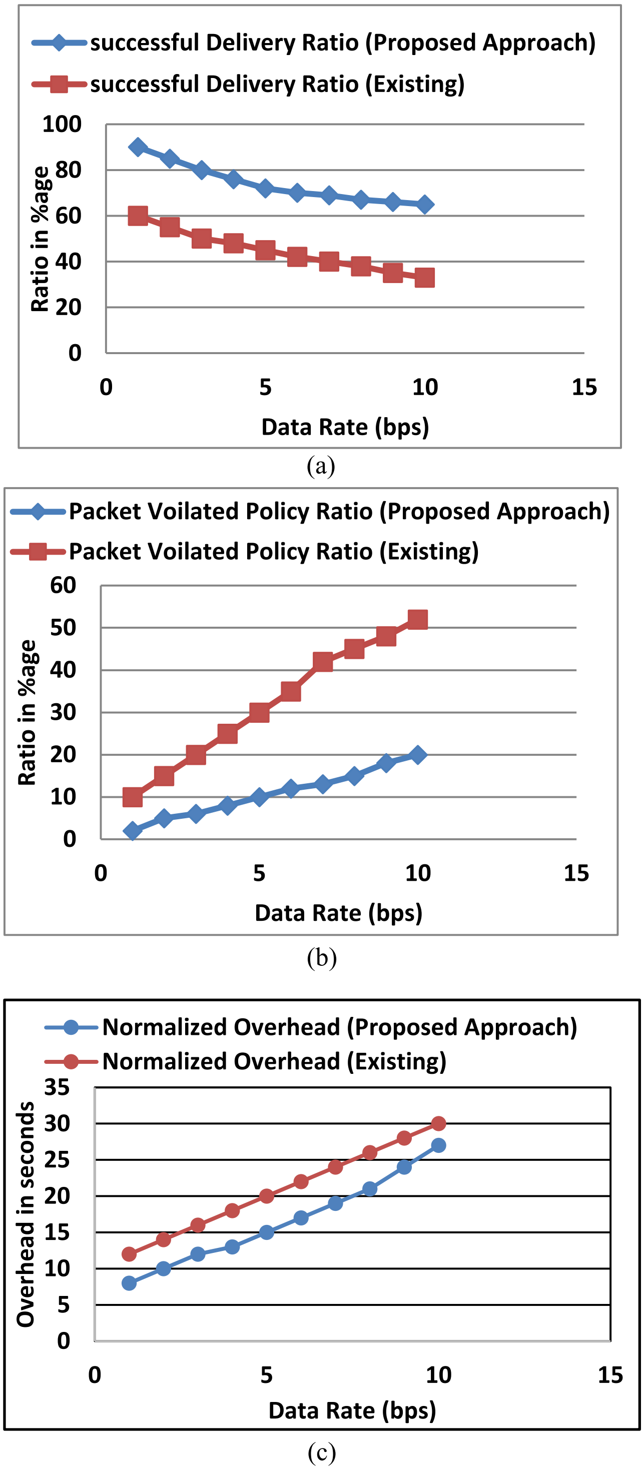


(a)

(b)

(c)

FIGURE 10. Result in case of No. of links changed. (a) Comparison of SPD Ratio. (b) Comparing both approaches in term of PPV. (c) Comparing normalized Overheads of both approaches.



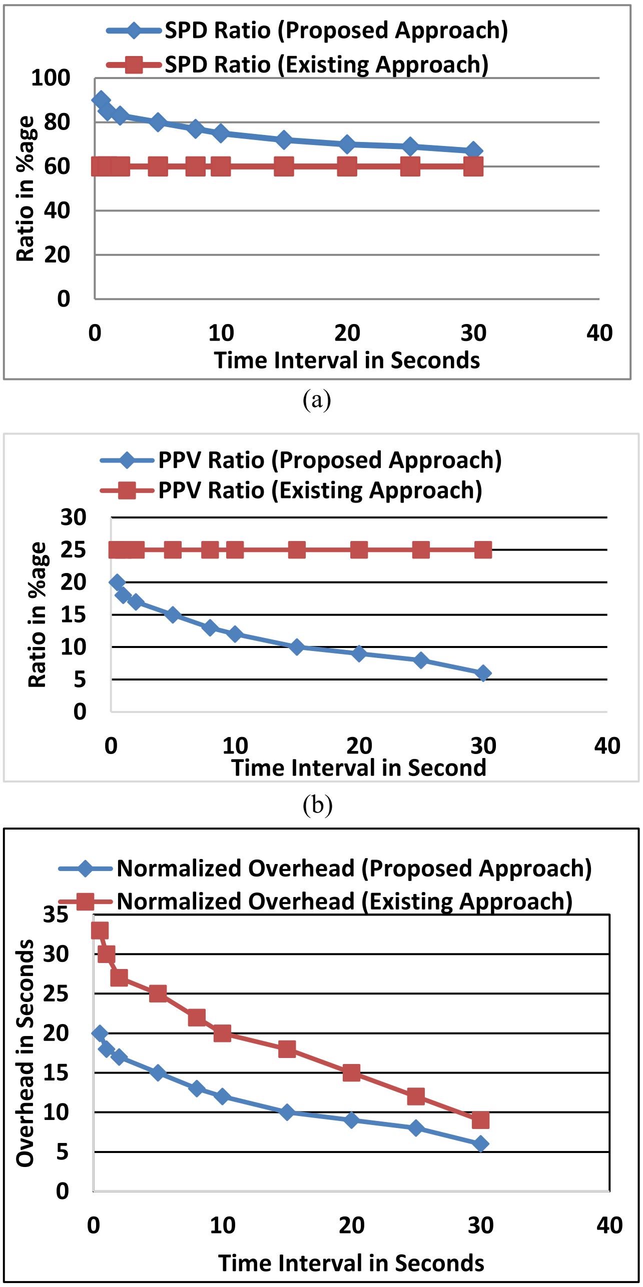
(a)

(b)

(c)

FIGURE 11. Effect of Data rate on SPD, PPV and Normalized Overhead. (a) Comparing SPD in case different data rates. (b) Effect of Data rate on PPV in both schemes. (c) Normalized Overhead in case of varying data rate for both schemes.

our approach. Though our proposed approach incurs some traffic overhead by getting log files periodically from the legacy forwarding devices for network topology (graph) construction. However, this traffic overhead is amortized by the gain in both SPD as follows. From Figure 10(c) one can notice that our proposed approach improves the performance of the network by having a lower value of normalized overhead as compared to existing approach. Our proposed approach also decreases the network administrator interaction with the



(a)

(b)

(c)

FIGURE 12. Change in network topology computation and its effect. (a) SPD ratio with varying time interval. (b) PPV ratio with different time intervals. (c) Normalized Overhead in case of time interval for topology change.

network to manually detect and address ACL policy violation in the case of topology change. Because our proposed approach automatically detects ACL policy violation due to topology change and helps a network administrator to configures that ACL policy in a better way.

B. RESULTS BY VARYING DATA RATES

For these results, we keep the link/failure value on average as 3, a number of nodes are 10 and the number of switches are 5 and the time interval for topology computation is 10 seconds. By varying sending data rates from 1 to 10 bits per seconds (bps), Figure 10 shows the effect of the data rate on SPD ratio, PPV ratio, and normalized overhead.

Figure 11 (a) indicates that SPD ratio of both the approaches decreases with the increase in data rates due to following reason. Due to higher data rate, when the network topology is changed then a large number data packets will be reaching on the faulty interfaces during the process of detection and implementation of the policy violation. This situation will result in traffic loss of most of the packets. However, our approach has higher SPD ration as compared to existing one.

It is indicated in Figure 11 (c) that normalized overhead gets increased in both approaches as data rates get increased. This is because a higher number of packets violate the policy in case of higher data rates as shown in Figure 10 (b). Figure 11 (c) shows that normalized overhead in our approach still is acceptable even in the case of increased data rate.

C. RESULTS BY VARYING TIME INTERVAL FOR TOPOLOGY CHANGE DETECTION

To get the results for a time interval, we used some simulation parameter as constant. we kept the link failure value on average as 3, a number of nodes are 10, the number of switches are 5 and data rate is 5 kbps (Kilobits per seconds). In our proposed approach, we get the link state information from forwarding devices after each time interval, say t, in order to detect the change in network topology. Results in Figure 12 show that SPD ratio, normalized overhead, and PPV ratio decreases with the increased value of t. The reason for this is explained as follows.

For a large value of t, it has low traffic overhead because less number of messages are sent for link state information in our approach. However, this cannot detect the change in network topology at an earlier stage. Similarly, by having a smaller value of t, there is more computation and traffic overhead in our approach. But this can detect a change in the network topology at an earlier stage. Subsequently, this results in enforcing the policy at an early time after traversing the tree as per our proposed approach.

However, our approach outperforms the existing approach in all cases as shown in Figure 12.

In Figure 12 (c) it is indicated that when time interval to get topology information is longer than traffic overhead is less and vice versa. This is because of the computation of topology information and detection of a change in network topology.

VI. CONCLUSION

We proposed in this paper a new approach for Hybrid SDN that auto-detects the interfaces of forwarding devices and network policies that are affected due to change in network topology. In the proposed approach, we modeled network-wide policy and local policy at forwarding device using a 3-tuple and a 6-tuple, respectively. We computed graph to represent the topology of the network.

By using Graph Difference, we detected a possible change in topology. In the case of topology change, we verified the policy for updated topology by traversing the tree using 6-tuple. If there was any violation in policy implementation, then affected interfaces were indicated and policies that need to be configured were also indicated. Then policies were configured on the updated topology according to specification in an improved way. Simulation results showed that our proposed approach outperforms the existing approach in term of successful packet delivery ratio, the ratio of packets that violated the policy and normalized overhead. We would like to consider more complex network policies violation as future work.

REFERENCES

[1] D. Levin *et al.*, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX Annu. Techn. Conf. (USENIX ATC)*, 2014, pp. 333–345.

[2] Z. Kerravala, "Configuration management delivers business resiliency," Yankee Group., Boston, MA, USA, Tech. Rep., Nov. 2002.

[3] M. Markovitch and S. Schmid, "SHEAR: A highly available and flexible network architecture marrying distributed and logically centralized control planes," in *Proc. 23rd IEEE ICNP*, Nov. 2015, pp. 78–89.

[4] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proc. Present. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2013, pp. 15–27.

[5] C. Prakash *et al.*, "Pga: Using graphs to express and automatically reconcile network policies," in *Proc. ACM Conf. Special Interest Group Data Commun.*, Aug. 2015, pp. 29–42.

[6] H. Lu, N. Arora, H. Zhang, C. Lumezanu, J. Rhee, and G. Jiang, "Hybnet: Network manager for a hybrid network infrastructure," in *Proc. Ind. Track 13th ACM/IFIP/USENIX Int. Middleware Conf.*, 2013, Art. no. 6.

[7] T. Feng and J. Bi, "OpenRouteFlow: Enable legacy router as a software-defined routing service for hybrid SDN," in *Proc. 24th Int. Conf. IEEE Comput. Commun. Netw. (ICCCN)*, Aug. 2015, pp. 1–8.

[8] S.-Y. Wang, C.-C. Wu, and C.-L. Chou, "Constructing an optimal spanning tree over a hybrid network with SDN and legacy switches," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2015, pp. 502–507.

[9] Y. Ganjali and N. McKeown, "Routing in a highly dynamic topology," in *Proc. SECON*, Sep. 2005, pp. 164–175.

[10] *Mininet*, accessed on Dec. 3, 2016. [Online]. Available: http://mininet.org/

[11] A. Leman, H. Tong, and D. Koutra, "Graph based anomaly detection and description: A survey," *Data Mining Knowl.*, vol. 29, no. 3, pp. 626–688, May 2015.

[12] R. Hand and E. Keller, "ClosedFlow: Openflow-like control over proprietary devices," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Aug. 2014, pp. 7–12.

[13] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Apr. 2012, p. 9.

[14] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Commun. Mag.*, vol. 2, no. 53, pp. 90–97, Feb. 2015.

[15] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: Toward an open-source solution for cloud computing," *Int. J. Comput. Appl.*, vol. 55, no. 3, pp. 38–42, Oct. 2012.

[16] R. Kumar, "Open source solution for cloud computing platform using OpenStack," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 5, pp. 89–98, 2014.

[17] N. Yee-Man, "Customer relationship management and recent developments," *Administ. Sci.*, vol. 6, no. 3, p. 7, 2016.

[18] C. Jin, C. Lumezanu, Q. Xu, Z.-L. Zhang, and G. Jiang, "Telekinesis: Controlling legacy switch routing with openflow in hybrid networks," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015.

[19] T. Nelson, A. D. Ferguson, D. Yu, R. Fonseca, and S. Krishnamurthi, "Exodus: Toward automatic migration of enterprise network configurations to SDNs," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, pp. 1–7.

[20] T. Mizrahi, E. Saat, and Y. Moses, "Timed consistent network updates," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, 2015, Art. no. 21.

[21] B. Arsić, D. Cvetković, S. K. Simić, and M. Škarić, "Graph spectral techniques in computer sciences," *Appl. Anal. Discrete Math.*, vol. 6, no. 1, pp. 1–30, Apr. 2012.

[22] S. Sarkar and P. Soundararajan, "Supervised learning of large perceptual organization: Graph spectral partitioning and learning automata," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 5, pp. 504–525, May 2000.

[23] W. Fan, X. Wang, and Y. Wu, "Incremental graph pattern matching," *ACM Trans. Database Syst.*, vol. 38, no. 3, Aug. 2013, Art. no. 18.

[24] *POX*, accessed on Dec. 3, 2016. [Online]. Available: https://github.com/noxrepo/pox

[25] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 70–75, 2014.

[26] M. Casado *et al.*, "Rethinking enterprise network control," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1270–1283, Apr. 2009.

[27] L. Gupta, R. Jain, and G. Vaszkun, "Survey of important issues in UAV communication networks," *IEEE Commun. Surveys Tut.*, vol. 18, no. 2, pp. 1123–1152, 2nd Quart., 2015.

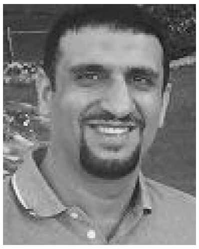
[28] A. R. D. Koutra, A. Parikh, and J. Xiang, "Algorithms for graph similarity and subgraph matching," Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep., 2011. [Online]. Available: https://pdfs.semanticscholar.org/5055/1518a193ffe3f3155368b43ba27ebc9c194b.pdf

[29] S. Vissicchiom, L. Vanbever, and J. Rexford, "Sweet little lies: Fake topologies for flexible routing," in *Proc. 13th ACM Workshop Hot Topics Netw.*, 2014, p. 3.

[30] S. Gai and K. McCloghrie, "Method and apparatus for defining and implementing high-level quality of service policies in computer networks," U.S. Patent 6 167 445, Dec. 1, 2000.

RASHID AMIN received the M.S. degree in computer science and the master's degree in computer science from International Islamic University, Islamabad. He is currently pursuing the Ph.D. degree with the COMSATS Institute of Information Technology, Wah Cantt, Pakistan. His M.S. thesis was on peer-to-peer overlay network over mobile ad hoc network. He was a Lecturer with the University of Wah, Wah Cantt, for four years. He has been a Lecturer with the Department of Computer Science, University of Engineering and Technology, Taxila, Pakistan, since 2014. His area of research is software-defined networking (SDN), under the supervision of Dr. N. Shah, hybrid SDN, P2P, and ad hoc network. He has been serving as a Reviewer for international Journals, including the *Journal of Network and Computer Applications*, the *Peer-to-Peer Networking and Applications*, and the *Frontiers of Computer Science*.

NADIR SHAH received the B.Sc. and M.Sc. degrees from Peshawar University, Peshawar, Pakistan, in 2002 and 2005, respectively, the M.S. degree from International Islamic University, Islamabad, Pakistan, in 2007, all in computer science, and the Ph.D. degree from Sino-German Joint Software Institute, Beihang University, Beijing, China. He was a Lecturer with the Department of Computer Science, COMSATS Institute of Information Technology, Abbottabad, Pakistan, from 2007 to 2008. He is currently an Associate Professor with the COMSATS Institute of Information Technology. He has authored several research papers in international journals/conferences, such as the *ACM Computing Surveys* and the IEEE Communication Letters. His current research interests include computer networks, distributed systems, and network security. He has been serving as a Reviewer for several journals/conferences, including the ICC, the INFOCOM, the WCNC, *Computer Networks* (Elsevier), the IEEE Communications Letters, the *IEEE Communication Magazine*, the IEEE Transactions on Industrial Informatics, and *The Computer Journal*.



BABAR SHAH received the two master's degrees, the master's degree in computer networks from Derby University, U.K. in 2007, and the master's degree in computer science from Peshawar University, Pakistan, in 2002, and the Ph.D. degree on the topic of Energy Efficient Wireless and Mobile Communication from Gyeongsang National University, where he studied with the Department of Informatics. He was with an academic and industry in South Korea, Oman, U.K., Rep of Ireland, and Pakistan. He is currently an Assistant Professor with the College of Information Technology, Zayed University. His research interests include center on improving the energy efficiency in wireless and mobile networks, secure routing protocols for both WSNs and MANETs, searching in peer-to-peer networks, and communication in 3-D WSNs, and the other aspects of informatics. He has authored many articles in this area.

OMAR ALFANDI received the M.Sc. degree in telecommunication engineering from the Kaiserslautern University of Technology, Germany, in 2005, and the Ph.D. degree (Dr.rer.nat.) in computer engineering and telematics from the Georg-August-Universität Göttingen, Germany, in 2009. He carried his Doctoral Research as part of an Industry, Academia, and Research centers collaboration European Union (EU) Project. He was a Package Leader of EU DAIDALOS II in the sixth framework project. He founded a Research and Education Sensor Laboratory, where he is currently as Lab Advisor. He is the Founder and the Director of the Sensors and Mobile Applications Research and Education Laboratory with CTI. He is currently an Associate Professor with the College of Technological Innovation, Zayed University, and an Adjunct Faculty with the Georg-August-University of Goettingen. He has authored numerous articles on Authentication Framework for 4G Communication Networks, Future Internet and Trust and Reputation Systems in Mobile ad hoc and Sensor Networks. His current research activities are Internet of Things, security in next generation networks, smart technologies, security engineering, mobile and wireless communications. From 2009 to 2011, he received the Post-Doctoral Fellowship from the Telematics Research Group.