Detection of Flow Violation in Distributed SDN Controller

Biswajit Halder, Mridul S. Barik, and Chandan Mazumdar

Jadavpur University, Kolkata, INDIA
Email: {biswajit.appl, mridulsankar, chandan.mazumdar}@gmail.com

***Abstract*—Software Defined Network (SDN) paradigm has revolutionized the way enterprise networks are designed by separating the control and data plane. It introduces a programmable network architecture which enables rapid and open innovation in different network functions that are allowed to install flow rules in forwarding elements via protocols like OpenFlow. Packet processing also becomes easier due to availability of packet information across different layers. But all these benefits may turn into great challenges because of the use of some features in OpenFlow. One of them, the *set_field* feature is widely used by network functions like firewall, router, load balancer etc. to modify packet header while in transit. Like distributed firewall setup, where rule in one firewall may conflict with that of other, in SDN, if multiple controller is used for individual networks or subnetwork within an organization, then change in flow rule in one controller may conflict with flow rules in other controller. Un-monitored management of flow may cause packets to loop through switches in single or multiple network, adversely affecting the network performance. In this paper, we introduce a mechanism using a directed graph representation to detect forwarding rules that cause forwarding loop, direct or indirect flow violation in a distributed controller environment. This helps network administrators to avoid possible security breaches, network congestion or even complete network failure caused by misconfiguration in security policies in different subnetworks.**

***Index Terms*—Software-Defined Network(SDN), Firewall, Directed Graph, Flow rule Violation, Forwarding loop, Security**

I. Introduction

Software-Defined Networking (SDN) [5] is a new network paradigm that has left great impact in implementation of enterprise network since last few years. SDN introduced rapid programmability by centralizing control logic and separating control plane from data plane, that are otherwise placed together in traditional network devices such as routers, switches etc. Centralized control and global view of the network helps easy deployment of different security devices and/or applications. SDN makes network management better than that of traditional network and makes implementation easier by running different security mechanisms like firewall, IDS, IPS etc. and network functionalities like routing, load balancing as controller applications. In a traditional network these functions are deployed as separate active devices. Beside several benefits SDN introduces some new challenges too. In SDN many controller applications are authorized to install flow rules in SDN switches, but lack of adequate flow management makes SDN vulnerable. This may cause other problems like flow rule over-write, indirect flow violation, forwarding loop etc. Dynamic modification of flow rule by different controller applications can make network state inconsistent with respect to the overall security policy of the network. When multiple controller is used in a network, forwarding rules from one controller may directly or indirectly conflict with that of the other and even though there exist no direct or indirect flow violation and/or forwarding loop in individual controller, combination of rules from different controller may lead to flow rule violation and forwarding loop resulting in security breach, network congestion and even network failure.

In this paper, we propose a solution for real-time detection of conflicts in distributed SDN controller environment. This approach involves generation of directed graph from the current forwarding rule set of firewalls, and that of other applications. NAT or PAT table of each controller are also considered. From this directed graph we show how to detect any direct or indirect flow rule violation and formation of forwarding loops in distributed controller environment. In section II we briefly sketch the related works, section III shows basic working principle of SDN and issues that we have tried to solve followed by proposed solution and performance analysis in section IV. And finally we conclude our work in section V.

II. Related Work

In recent years SDN has become one of the most researched topic in the field of computer network. Researchers have proposed many ideas to solve SDN specific issues like controller and application fault tolerance, firewall mis-configuration, flow conflict, forwarding loop, performance optimization etc. Based on goals of research outcome that are related to our present work, we have categorized these works in three different groups.

- **Anomaly Detection:** Ehab S. Al-Shaer and H. Hamed [2] proposed a user friendly framework for automatic discovery of policy anomalies in legacy firewalls, and also for anomaly-free policy editing for rule insertion, removal and modification. The same authors in another work [3] presented a way to discover inter firewall anomalies. Jeffrey and Samak [6] presented a model checking technique using Binary Decision Diagram(BDD) and Boolean satisfiability (SAT) formulae for analyzing the behavior of firewall policy configurations, and anomaly detection.

- **Detection of Flow Rule Violation:** FlowChecker [4] is a tool to verify misconfigurations of a single FlowTable of a switch. FortNOX [10] is another security mediation service that deals with flow (firewall policy) misconfiguration. FLOVER[12] is another solution to verify compliance of dynamically assigned rules with invariant security policy in real time. Header Space Analysis (HSA) [7] is a mechanism based on analysis of packet header to detect forwarding loop in traditional network. In another work D. Kordalewski and R. Robere [8] has used HSA along with directed graph for detection of intra-network forwarding loop in SDN environment.
- **Firewall Performance Optimization** A. Tapdiya and E. W. Fulp. [13] proposed a solution for firewall performance optimization and integrity. In this work they addressed reordering of a firewall rule set to minimize the average number of comparisons to determine the action, while maintaining the integrity of the original policy.

Configuration of security applications individually that belongs to different controller is time consuming and erroneous, since change in configuration in policy of one controller application may conflict with that of other causing the whole network vulnerable. Conflict detection and removal in real time among different controller is a major task to do to secure the network in a distributed controller architecture. In this paper we have introduced a graph based solution to detect such conflict in real time to prevent any kind of flow related security breaches. In our approach each controller generates a directed graph based on our proposed algorithm from available rule sets of all security applications. Graphs from different controller are merged to generate a combined graph showing global network state. From this combined graph it is very easy to detect any kind of forwarding loop and/or flow violation.

III. Basic Working Principle

A. Flow installation in SDN Environment

In SDN, switches are dumb and their task is to forward packets as per instruction received from controller i.e. the control plane. Each switch forwards packet based on the flow rules stored in it. Flow rules are nothing but application rules translated in switch (data plane) understandable format. Each flow rule contains an action field, which specifies the operation(s) to be performed on the matched packet and the switch port through which it should be forwarded after execution of the said operation(s). On receiving a packet if the switch does not find any matching rule in its flow table, the packet information is forwarded to the SDN controller, where it passes through one or more applications and finally one flow rule corresponding to the packet is installed by the controller and associated action is executed on the packet. On receiving a packet if a matching rule is found then the corresponding action(s) is/are executed on the packet immediately without communication to the controller.

B. Flow Conflicts

OpenFlow [9] introduces some new challenges which are not seen in traditional networks. It introduces *set_field* option that allows modification of packet header by forwarding elements while passing through it. This option is very useful for dynamic routing and other redirection applications. To implement NAT/PAT (Network/Port Address Translation) in SDN, routing applications need to modify packet header content and the *set_field* option becomes handy in this scenarios. In todays network, where dynamic and automated control is very important, this option plays a major role. For example, based on traffic statistic, instead of delivering a packet directly, packet header can be modified and before delivering it to the intended receiver the packet can be made to pass through several security applications. DefenseFlow [11] is an example of such an application that uses packet redirection mechanism to prevent DDoS attacks. Moreover, instead of using a separate proxy server, administrators can use OpenFlow based switches themselves to act as proxy server by using *set_field* option. But all this advantages of OpenFlow comes with problems like *forwarding loop* and *direct or indirect flow violation*. These problems can occur either in a single controller or in multiple(distributed) controller scenario.

Table I show set of forwarding rules for each controller in the example network of Figure 1. For our test cases we have taken four networks with single controller in each network as shown in Figure 1. For simplicity we have assumed that, for these rules only the source and destination addresses are modified, and other fields like source port, destination port, ttl etc are not modified, hence are not shown in the resulting graph. In the graph store, each node contains all the fields (as node property) mentioned in a forwarding rule irrespective of whether it is modified or not. However, for clarity in presentation, in Figure 4 and 5 we have shown only the fields which are modified. Our, proposed method is capable of handling any number of field modification for a rule. In SDN there is no physical router, and routing is typically performed by a controller application. Here, any switch port can act as router interface(LAN/WAN). Therefore, a network can have multiple LAN/WAN interface in multiple switches as shown in our sample network (Figure 1). Network architecture in this figure is different from traditional network architecture. It is based on the routing concept mentioned above that illustrates the problem scenario clearly.

1) Multi Controller Forwarding Loop: Besides allow or drop options OpenFlow introduces many other actions related to packet forwarding. Source address, destination address, source port, destination port and many other layer 2/3/4 header fields can be modified using *set_field* action while it passes through OpenFlow enabled devices. In many cases, to cater to present needs, administrators or applications add forwarding rules without examining the current rule set. This can create single-controller or multi-controller forwarding loops and/or indirect flow violation.

Normally, switches forward a packet to a particular port

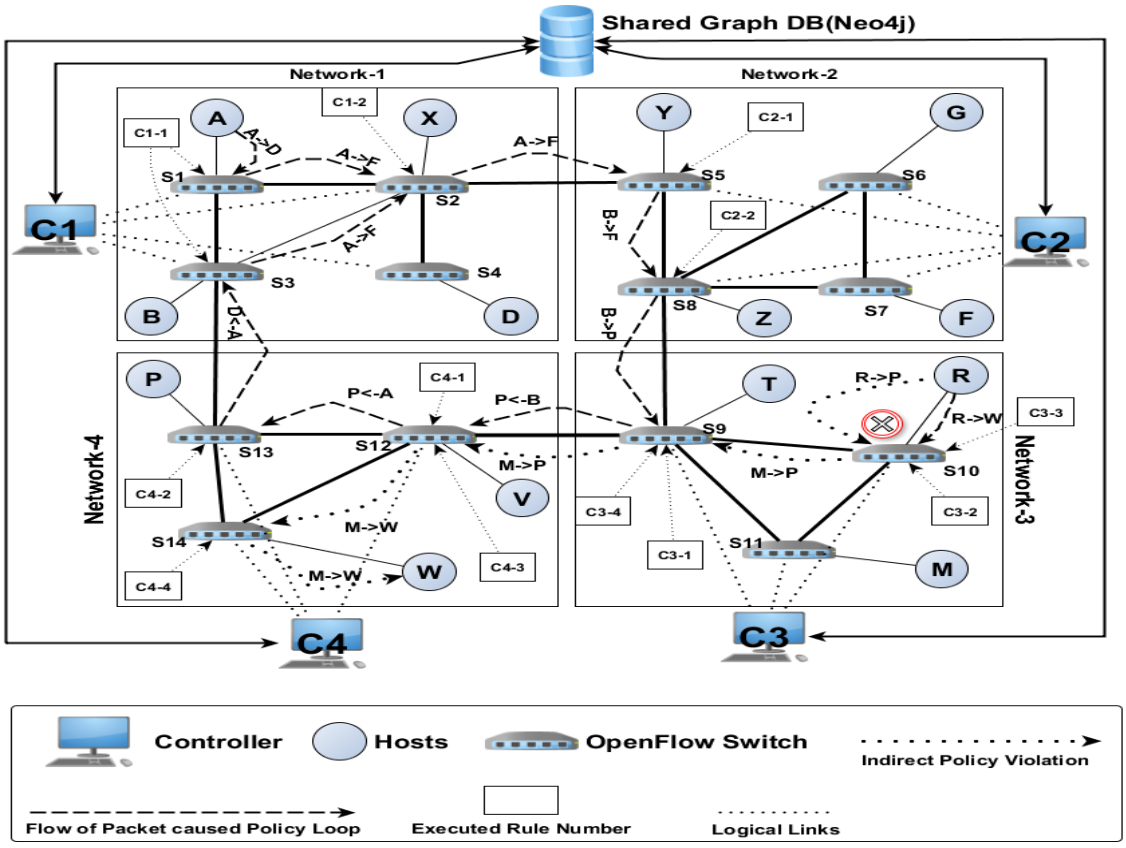

Fig. 1. Network Topology-1: Forwarding Loop & Indirect Violation

only after executing matched flow rule's actions on the packet. It decides the output port based on MAC-to-port information which is pre-computed based on the shortest path to the destination. From Figure 1 it is clear that none of the individual network contains a forwarding loop but rule sequence C1-1, C1-2, C2-1, C2-2, C2-3, C3-1, C4-1 and C4-2 from all the controller together does. If a packet is sent to host D from host A, it passes through network 2, network 3, Network 4 and finally returns to network 1, thereby creating a loop. Each OpenFlow devices in the path of the packet execute flow rule related to the packet applicable for that network. Execution of corresponding rule is shown in small rectangles connected by a dotted (- - →) arrow to the switch where it is executed. Too many packets from source A to destination D will cause congestion in the network.

2) Inter-Controller Indirect Flow Violation: Indirect flow violation occurs when final action executed by a set of forwarding rules together conflicts with that of another forwarding rule. Figure 1 shows inter-controller indirect flow violation. Rule C3-2 says that a packet from R to W should be dropped. But when a packet is sent from host R to host P, rule C3-3 is executed on switch-10 and the packet is forwarded after modifying the source address to M. Then rule C3-4 is executed on switch-9 and the packet is forwarded to the destination network. In network-4 rule C4-3 is executed on switch-12 and the packet is forwarded after modifying the destination address to W. After that rule C4-4 is executed on switch-14 and the packet is forwarded to host W which is a clear violation of rule C3-2.

TABLE I
SAMPLE FORWARDING RULE SET(FROM DIFFERENT APPLICATIONS AND NAT/PAT TABLE) IN DIFFERENT CONTROLLER

		<—Rule—>	
Contr-oller-#	Rule #	Src → Dst	Action
1	C1-1	A → D	Set destination address to F, Forward
	C1-2	A → F	Forward
	C1-3	X → F	Set source address to A, Forward
	C1-4	X → A	Forward.
	C1-5	ANY → ANY	Forward.
2	C2-1	A → F	Set source address to B, Forward
	C2-2	B → F	Set destination address to P, Forward
	C2-3	B → P	Forward
	C2-4	G → M	Forward
	C2-5	ANY → ANY	Forward
3	C3-1	B → P	Forward
	C3-2	R → W	Drop
	C3-3	R → P	Set source address to M, Forward
	C3-4	M → P	Forward
	C3-5	G → M	Drop.
	C3-6	ANY → ANY	Forward
4	C4-1	B → P	Set source address to A, Forward
	C4-2	A → P	Set destination address to D, Forward
	C4-3	M → P	Set destination address to W, Forward
	C4-4	M → W	Forward
	C4-5	ANY → ANY	Forward

In Table-I, the notation Ci-j indicates jth rule of ith controller, i.e. C1-2 indicates 2nd rule of controller 1.

IV. Proposed Solution

In this section, we propose a solution to the problems discussed in section III-B. It first generates a directed graph from the forwarding rules of individual controllers and then this graph is shared with other controllers by storing them in a shared graph database (we have used Neo4j[1] graph DB). The shared combined graph, generated from shared individual graph is then analyzed for detection of inter-network forwarding loop and inter-network indirect flow violation.

A. Graph Data Model

In our representation each graph node and edge satisfy the following properties:

- Each node in the generated graph (except two designated nodes 'DROP' and 'Forward'), its outgoing edge and target node of the edge together represents a forwarding rule.
- Each node contains all the field (except action) values of the concerned forwarding rule. If a rule is written based on source address, destination address, source port, destination port and protocol, every node contains a field related to all of these fields.
- Destination of each directed edge represent next state (i.e. whether the packet should be forwarded, should be dropped or should be modified.) If the target of the edge is 'Drop' node then the packet should be dropped, if the target is 'Forward' node then the packet need to be forwarded without any modification and if the target is any node other than 'Forward' or 'Drop' then the packet will be forwarded after header modification.
- Each edge has a field named "c_id" which indicates the controller to which the corresponding forwarding rule belongs.

B. Graph Generation

The graph generation process includes three steps that are repeated for each forwarding rule and for each entry in NAT or PAT table.

For Forwarding Rule

1) Add node, say N1 (if not exist) as per information available in forwarding rule, excluding the action field.
2) Add node, say N2 (if not exist) based on modified content as per action.
3) Add an edge from N1 to N2. Edge properties contain a field that stores the controller id to which the rule belongs. But if Destination of node N1 is outside network and action is either forward or drop then no need to add an edge as corresponding edge and nodes should have been added from NAT/PAT Table.

For better understanding of the above mentioned steps consider the below rules in Controller-1 (Controller ID:C1):

Rule-1: A $\rightarrow$ B, Src_port=80, Dst_Port=5555 and protocol=ICMP ACTION: set source address=D, set destination port=6600

Rule-2: A $\rightarrow$ B, Src_port=80, Dst_Port=80 and protocol=ICMP ACTION:Forward

Nodes related to Rule-1 and Rule-2 are shown in Figure 2.

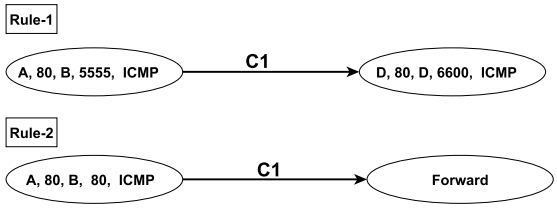

Fig. 2. Sample graph for single rule

For NAT or PAT Table

1) Add node, say N1 (if not exist) as per information available in R_src, R_sport, R_dst and R_dport.
2) Add node, say N2 (if not exist) based on T_src, T_sport, T_dst and T_dport. Here, R means received and T means Transmitted or Translated.
3) Add an edge from N1 to N2. Edge properties contains a field that stores the controller id to which the NAT or PAT table belongs.

For example consider the below PAT table (Table-II) of controller-1.

TABLE II
PAT Table of Controller-1 (Controller ID - C1)

R_src	R_sport	R_dst	R_dport	T_Src	T_sport	T_Dst	T_dport
A	5555	X	80	R	3000	X	80
X	80	R	3000	X	80	A	5555

Equivalent forwarding rule for the PAT table mentioned in Table-II is given in Table-III

TABLE III
Forwarding rule for NAT/PAT table entry in Table-II

Contr-oller-#	Rule #	SrcIP:Src Port $\rightarrow$ DstIP:Dst Port	Action
1	C1-N1	A:5555 $\rightarrow$ D:80	Set source address to R, set source port to 3000, Forward
1	C1-N2	A:80 $\rightarrow$ R:3000	Set destination address to A, set destination port to 5555, Forward

Corresponding graph for the above two entries is shown in Figure 3.

Directed graph generated using above mentioned procedure for each controller (as per rules given in Table-I) is given in Figure 4 and the combined graph is shown in Figure 5. For better understandability, we have used different types of directed edges to show possible violations among forwarding rules and the fields that are not modified are not shown in the graph.

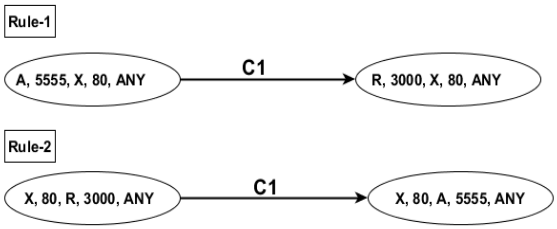

Fig. 3. Sample graph for PAT table(Table II/III) entry

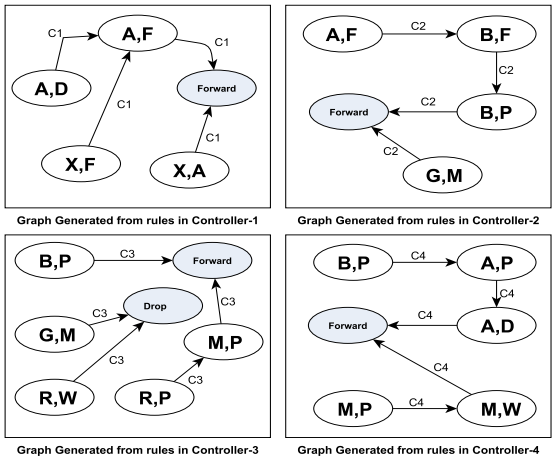

Fig. 4. Directed graph for individual firewall

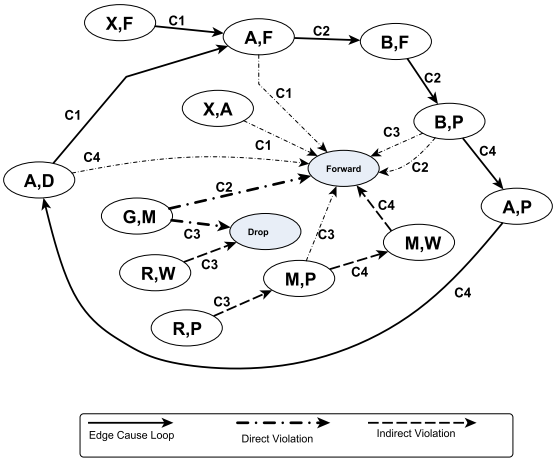

Fig. 5. Combined Directed graph with different types of arrow

C. Conflict Detection

For graph processing and conflict detection we have introduced two different algorithms - Algorithm IndirectFlowViolation(G_c) - finds out whether addition of a new rule will cause indirect Flow violation or not. G_c is the combined graph generated by merging individual controller graphs. Algorithm ForwardingLoop(G_c) finds whether addition of a rule R_x will cause an inter-controller and/or intra-controller loop or not. Both IndirectFlowViolation(G_c) and ForwardingLoop(G_c) takes two input, a graph G_c and the rule R_x that we want to insert.

For detecting indirect flow violation corresponding to rule R_x, represented by node N, we use IndirectFlowViolation(G_c) on the combined graph $G_c = (V, E)$. Where V=Set of vertex in G_c and E=Set of edges in G_c.

Algorithm 1: IndirectFlowViolation(G_c)

input : Directed Graph(G_c) and the rule R_x
output: set of rules that indirectly violate rule R_x

1 VRS_x = {}
2 Find the node N=$v_x^{in} \in V$.
3 Find a node (if exist) L$\in V$ such that $L_{TP-proto}$=$N_{TP-proto}$, L_{Src-ip}=N_{Src-ip}, $L_{Src-port}$=$N_{Src-port}$ and L_{Dst-ip}≠N_{Dst-ip}
4 Find a node U$\in V$(if exist) in the forwarding path from node L such that U_{Dst-ip}=N_{Dst-ip} and $L_{Dst-port}$=$N_{Drc-port}$
5 **if** *T*(e_N^{out})≠*T*(e_U^{out}) **then**
6 There is indirect violation for rule R_x
7 Let $e_i = e_N^{out} and e_j = e_U^{out}$
8 **if** $id(e_i) == id(e_j)$ **then**
9 This is a intra-controller indirect flow violation for forwarding rule R_x
10 **else**
11 This is a inter-controller indirect flow violation for forwarding rule R_x
12 P=Set of path from L to U→Next;
13 **for** *all nodes K in path* P_i *(*$P_i \in P$ *indicates* i^{th} *path)* **do**
14 $R_s = f(K)$; $VRS_x \leftarrow R_s$;
15 **else**
16 There is no indirect violation for rule R_x

Symbols used in algorithm IndirectFlowViolation(G_c) are as follows:

- VRS_x = Violated rule set for rule R_x, i.e. the set of rules that together violates rule R_x.
- R_i is the i^{th} forwarding rule.
- v_i^{in} is the node that represent the header content of the packet that matches rule R_i. Attributes of node v_i^{in} are same as the fields of the rules except the action field.
- N is a node that represent a forwarding rule in a graph. Fields of a forwarding rule is/are nothing but the node's attributes.
- $T(e)$ = target node of edge e.
- e_N^{out} = set of out going edges of node N.
- id(e) = id of edge e.
- U→Next = Successor node of node U.

Line 2 of algorithm IndirectFlowViolation(G_c) searches a matching node for rule R_x and it takes $O(V + E)$ time if either BFS or DFS is used. Here V is number of nodes and E is number of edges. Same thing applies for line 3 and line 4

with change in node properties. For line 4 it checks only a part of the graph, in worst case which is equal to the graph, leading to a time complexity of $O(V + E)$. Whereas in algorithm ForwardingLoop(G_c), edges(LP) in line 4 takes $O(E)$ time in worst case and that for nodes(LP^i) in line 5 is $O(V)$. Line 7 in algorithm ForwardingLoop(G_c) takes $O(V \times E)$ time which is the effective time complexity for the algorithm ForwardingLoop(G_c).

Algorithm 2: ForwardingLoop(G_c)

input : Combined Graph (G_c) and Rule R_x
output: Status of the loop

```
1  Let LP= findLoop(R_x). [Returns all loop that are created
     after insertion of rule R_x]
2  if LP is not NULL then
3      for each Loop LP^i ∈ LP do
4          E_LP^i = edges(LP^i) [Returns all the edges in the
             loop LP^i]
5          N_LP^i = nodes(LP^i) [Returns all nodes in the
             loop LP^i]
6          Flag=0
7          for each node X∈ N_LP^i do
8              Let e_X^in ∈ E_LP^i and e_X^out ∈ E_LP^i
9              if id(e_X^in) ≠ id(e_X^out) then
10                 if isEdge(X,v_f) then
11                     Flag=1; continue;
12                 else if isEdge(X,v_d) then
13                     Flag=2; break;
14                 else
15                     Flag=3; break;
16         if Flag==1 then
17             The loop LP^i is an inter-controller
                 forwarding loop
18         else if Flag==2 then
19             Forwarding loop will be created if forwarding
                 rule that represent the node X is executed at
                 a terminal-switch#
20         else if Flag==3 then
21             Though loop LP^i is present in G_c, the set of
                 rules present in loop LP will not create a
                 forwarding Loop
22         else
23             The loop LP^i is an intra-controller
                 forwarding loop
24 else
25     Insertion of rule R_x will not create any forwarding
         loop.
```

#terminal-switch: For a rule R_x terminal switch is that switch, where after execution of rule R_x, a packet leaves the current network.

Symbols used in algorithm ForwardingLoop(G_c) are as follows:

E is the set of all edges. isEdge(X, v_f) checks whether there is a directed edge from node X to node v_f or not. e_N^{in} represent set of incoming edges of node N. Rest of the symbols are are same as Algorithm 1.

V. Conclusion and future work

OpenFlow brings innovation in network development and makes a network programmable. But it also introduces options for modification in packet header contents that causes serious problem like flow violations. In this paper, we have introduced two algorithms, algorithm 1 detects any inter-controller and intra-controller indirect flow rule violation and algorithm 2 detects forwarding loop in both intra-controller and intra-controller Software Defined Network. As a future extension of this work we intend to use this mechanism for detection of flow violations and forwarding loops in global SDN environment using blockchain and graph databases.

VI. Acknowledgments

Authors would like to thank the Centre for Distributed Computing, Jadavpur University, for supporting this work. Also, we acknowledge the Council of Scientific and Industrial Research (CSIR), Govt. of India for funding the first author.

References

[1] Neo4j graph database, neo technology, inc. https://neo4j.com/, Jul 2017.
[2] E. S. Al-Shaer and H. H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, pages 17–30, March 2003.
[3] E. S. Al-Shaer and H. H. Hamed. Modeling and management of firewall policies. In *IEEE Transactions on Network and Service Management*, volume 1, pages 2–10, April 2004.
[4] Z. Chen, Q. Gao, W. Zhang, and F. Qin. Flowchecker: Detecting bugs in mpi libraries via message flow checking. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2010.
[5] N. Feamster, J. Rexford, and E. Zegura. The road to sdn: An intellectual history of programmable networks. volume 44, pages 87–98, New York, NY, USA, Apr. 2014. ACM.
[6] A. Jeffrey and T. Samak. Model checking firewall policy configurations. In *IEEE International Symposium on Policies for Distributed Systems and Networks, 2009.*, pages 60–67, July 2009.
[7] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 1–14, Berkeley, CA, USA, 2012. USENIX Association.
[8] D. Kordalewski and R. Robere. A dynamic algorithm for loop detection in software defined network. In *Technical Report, University of Toronto*, December 2012.
[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.
[10] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 121–126, New York, NY, USA, 2012. ACM.
[11] Radware. Ddos protection: Load balancing and application delivery solutions — radware. https://www.radware.com/, Feb 2017.
[12] S. Son, S. Shin, V. Yegneswaran, P. A. Porras, and G. Gu. Model checking invariant security properties in openflow. In *ICC*, pages 1974–1979. IEEE, 2013.
[13] A. Tapdiya and E. W. Fulp. Towards optimal firewall rule ordering utilizing directed acyclical graphs. In *Proceedings of 18th Internatonal Conference on Computer Communications and Networks, 2009. ICCCN 2009.*, pages 1–6, Aug 2009.