Computers, Materials & Continua
DOI: 10.32604/cmc.2023.031282
Article

Tech Science Press

Fast Verification of Network Configuration Updates

Jiangyuan Yao[1], Zheng Jiang[2], Kaiwen Zou[2], Shuhua Weng[1], Yaxin Li[3], Deshun Li[1], Yahui Li[4,*] and Xingcan Cao[5]

[1]Scool of Computer Science and Technology, Hainan University, Haikou, 570228, China
[2]School of Cyberspace Security, Hainan University, Haikou, 570228, China
[3]School of Software Engineering, Jilin University, Changchun, 130012, China
[4]School of Software, Beijing Jiaotong University, Beijing, 100091, China
[5]University of British Columbia, Vancouver, V5K1K5, Canada
*Corresponding Author: Yahui Li. Email: liyahui_ego@126.com
Received: 14 April 2022; Accepted: 10 June 2022

Abstract: With the expansion of network services, large-scale networks have progressively become common. The network status changes rapidly in response to customer needs and configuration changes, so network configuration changes are also very frequent. However, no matter what changes, the network must ensure the correct conditions, such as isolating tenants from each other or guaranteeing essential services. Once changes occur, it is necessary to verify the after-changed network. Whereas, for the verification of large-scale network configuration changes, many current verifiers show poor efficiency. In order to solve the problem of multiple global verifications caused by frequent updates of local configurations in large networks, we present a fast configuration updates verification tool, FastCUV, for distributed control planes. FastCUV aims to enhance the efficiency of distributed control plane verification for medium and large networks while ensuring correctness. This paper presents a method to determine the network range affected by the configuration change. We present a flow model and graph structure to facilitate the design of verification algorithms and speed up verification. Our scheme verifies the network area affected by obtaining the change of the Forwarding Information Base (FIB) before and after. FastCUV supports rich network attributes, meanwhile, has high efficiency and correctness performance. After experimental verification and result analysis, our method outperforms the state-of-the-art method to a certain extent.

Keywords: Network verification; configuration updates; network control plane; forwarding information base

1 Introduction

Nowadays, network configurations are changed and upgraded rapidly by network managers to make the network configured compatible with new devices or new protocols to expand the business and improve security [1–5]. In tier-1 Internet Server Provider (ISP), the upgrade with new-generation

devices appeared about 17 times in a year. And the times of configuration changes in Border Gate Protocol (BGP) sessions are more than 400 per month, however, quite a few of them are slight changes [6].

On October 4, 2021, Facebook's Instagram, Oculus, and other platforms suffered a service interruption that lasted more than seven hours globally, causing the company to directly lose more than 500 million yuan. This was a chain event that was caused by a configuration error and its main reason was Domain Name System (DNS) resolution failure caused by BGP. In 2012, the Microsoft Azure data center caused an unknown configuration error, which led to a decline of service quality in Western Europe for more than 2 hours. Between 2016 and 2017, 56% of Alibaba network errors also came from configuration changes. For these large-scale networks, the cost of global configuration verification is extremely high, so applying a fast and light configuration verifier is extremely crucial.

The development of network control plane verification technology has passed several years since Batfish [7] in 2015, but there is still no suitable control plane verification tool has been reported to handle the verification from medium to large networks at present. In the research process of network control plane verification, the network control plane verification faces two major challenges in accuracy and scalability issues.

The accuracy problem comes from the fact that some large-scale networks generally consist of devices from different vendors. On the one hand, there are grammatical differences between devices of different venders, which brings difficulties to the configuration analysis of different devices; on the other, even if the vendor's models with similar grammar or the vendors of different grammar are mapped to the same semantics, the same configuration expresses the behavior of network devices is not completely consistent, which leads to the existing of Vendor-Specific Behavior (VSB).

Because of the existence of differences between vendors, only using a set of network models to describe the entire network will cause differences between theoretical models and actual network behaviors, which will lead to the scalability issues of verification. Through the comparison of existing verification tools, we discovered that for large-scale network verification (e.g., hybrid multi-domain software-defined network [8]), most of the network control plane verification algorithms have scalability problems in certain verification scenarios.

These scalability problems come from the cost of a single verification for a certain autonomous system and the efficiency of network control plane verification caused by the huge network scale. If we rely on logical expressions to solve them, we will run into new problems. The computational cost of algorithms will be extremely high when the network scale is huge; and if simulation or model checking is applied, it will cause the problem of state explosion, too much simulation environment, and low efficiency.

Batfish can support the cleaning of configurations from multiple vendors and can provide FIB tables according to the current network configuration, which solves the VSB problem very well. It obtains the data plane through configuration files, calculates the FIB, and performs network attribute verification based on the control plane information. And for data plane verification, there are many verification tools, such as Anteater [9], Head Space Analysis (HSA) [10], Veriflow [11], and Atomic Predicates (AP) [12].

FIB exists in each device of the distributed network and maintains the forwarding information of the device. When the routing in the network changes, devices will calculate routing information and generate a new FIB based on the change. No matter how the configuration changes, the data packet will still be transmitted according to the next hop indicated by the after-changed FIB in the device.

Therefore, we start from this point to figure out the location of configuration changes by comparing the global FIB of the network before and after the change and then determine the range that is affected by the configuration change. Verify the affected network, or verify it based on the network operator's intentions.

This paper is structured as follows. Section 2 introduces the related work of control plane verification and data plane incremental verification. Section 3 introduces a motivating instance to illustrate the problem and then explains the opportunities for improvement. We propose the network model to address the network verification issues arising from the configuration update in Section 4. Section 5 determines the network area affected by configuration changes. We present FastCUV, a verifier for distributed network general attributes in Section 6. In Section 7, we conducted an experimental evaluation and analysis of the results. Section 8 introduces the discussions and our future work.

2 Related Work

In this section, we discuss the verification work about the control plane of the traditional network, the incremental verification works of the data plane, and the recent new methods of verification in Software Defined Network (SDN) respectively.

2.1 Control Plane Verification

There are four main types of study design used to accomplish verification of the control plane: simulation-based, model-checking-based, graph-based, and Satisfiability Modulo Theories (SMT)-based. In the following, we introduce some outstanding works of each type. Finally, we analyze the advantages and disadvantages of these works. The tools described in this section mainly include Batfish [7], Abstract Representation for Control planes (ARC) [13], Tiramisu [14], Minesweeper [15], FLOVER [16], Plankton [17], and FlowChecker [18]. They are listed in Tab. 1.

Table 1: Classical control plane verification tools

Simulation-based	Model-checking-based	Graph-based	SMT-based
Batfish	Plankton FlowChecker	ARC Tiramisu	Minesweeper FLOVER

2.1.1 Simulation-based Control Plane Verification Work

This kind of work simulates the network by generating network snapshots and calculating the FIBs of each device by inputting the configuration information and network environment. After simulation, network snapshots will be put into data plane verification tools and be verified in this way.

The most popular work in simulation-based verification is Batfish. Batfish can convert a multi-vendor configuration into a standard control plane information. By abstracting configuration information and topology information, Batfish can construct a control plane model, and then generates one or more data plane snapshots according to the attribute which will be verified. The model can give results or counter-examples based on the messages that users query.

Batfish pioneered the use of simulation combined with data plane verification for control plane verification. But its disadvantage is that the efficiency will drop sharply when generating multiple

network snapshots. Because there are too many environments for calculation, it can only be simulated separately for each environment and then verified. And detailed analysis of the overall network environment for every attribute is not necessary in many cases.

But the only convergent routing calculation result can be obtained, and the enumeration method is used for verification, which makes it too inefficient.

2.1.2 Model-checking-based Control Plane Verification Work

This technique checks whether the system model satisfies the specifications. Most work of this type accomplish the verification by modeling the network, defining the attributes, and utilizing the model checker to check out the violations.

Plankton's main contribution is the promotion of the scalability of existing works by merging equivalence partitioning and explicit state model checking. Plankton references Veriflow, reducing the packet header space that needs to be checked using equivalence division, and explicit-state model checking is used to resolve the problem of data plane diversity caused by different convergence states of the control plane. Nevertheless, plankton is still the same as batfish, using an enumeration to check the failure scenarios in the state space.

FlowChecker is used to perform verification to SDN. It models forwarding rules to Boolean expressions and uses Computation Tree Logic (CTL) to express network invariants. But it suffers from low scalability issues and has state space explosion risks.

2.1.3 Graph-based Control Plane Verification Work

The graph-based methods mainly include ARC and Tiramisu. They simulate the interaction of network devices and protocols through a graph model. The vertex in the graph represents the device or port, the link and the cost are represented by the edge and the weight.

ARC uses batfish as the configuration parser to convert configuration files into weighted directed graphs. The work of ARC matches the weight of the redistribution between OSPF and BGP. ARC also solves the problem of setting the abstract edge weight of the graph model. This problem arises from the different weight measurements of the interior gateway protocol and the exterior gateway protocol.

Tiramisu is more like an extension of ARC's work. Tiramisu has designed different solution models for different verification scenarios. Different solutions can be given to different problems so that Tiramisu can achieve high efficiency, which is its biggest advantage. The establishment of graphs is relatively fast and the complexity of graph algorithms is rather low. The disadvantage of it is that too many configuration details are ignored in the abstraction, resulting in unsupported many network attributes for verification and its poor scalability.

2.1.4 SMT-based Control Plane Verification Work

Minesweeper is a representative work based on SMT solutions. It abstracts the entire network model and has a high network design coverage. Therefore, it can support the verification of rich attributes. It encodes the entire network as SMT formulas, which describe the routing and protocol exchange process in detail. But because it is based on SMT, its solution time will increase with the expansion of the network scale, so the verification effect for simple attributes of large networks who's update with small changes is not satisfactory. And FLOVER uses SMT to solve the verification problem of the OpenFlow protocol in SDN.

2.2 Data Plane Incremental Verification

The incremental verification for the data plane refers to the partial verification of the relevant attributes of the data plane by monitoring the changes of the data plane in real-time or by comparing changes of the data plane during the previous verification and current verification. We will clarify this issue by introducing Netplumber [19], VeriFlow, and Atomic Predicates (AP) Verifier.

Netplumber implements real-time incremental network data plane verification based on HSA. Netplumber checks the compliance of state changes incrementally. It models the network box as a node and establishes a dependency graph between rules, that is, a Rule Dependency Graph (RDG). The network invariants are equivalently converted into reachability assertions. Once the network changes (the message goes through the network box), it will update the corresponding network dependency graph and re-do all forms of verification. Once a breach is discovered, NetPlumber will block the change. Changes must be determined by rules, and adding rules will not significantly affect the equivalence class of forwarding rules.

Nevertheless, NetPlumber takes a long time to process link access and is not suitable for networks with frequent link changes. VeriFlow and NetPlumber achieve similar runtime performance. Similar to VeriFlow, NetPlumber is protocol-independent and can additionally verify any header modification, including rewriting and encapsulation.

The AP is more efficient than previous tools, such as NetPlumber and Veriflow, for network verification. The packet filter is represented as a set of predicates. It developed a new algorithm setting for calculating atomic predicates. Each predicate can be expressed as a disjunctive atomic predicate, which speeds up the calculation. Atomic predicates can be stored as integers, and disjunctions are calculated as the union of integer sets. Therefore, the data packet set can be swiftly calculated, and minimum the atomic predicate so that eliminating the redundancy in the forwarding and Access Control List (ACL) to pass the AP verification.

The verification of the data plane can guarantee whether the data plane in the current state is safe and whether it satisfies some established attributes. The input of Netplumber is the Layer 2 (L2) information, such as port, Virtual Local Area Network (VLAN), etc., and real-time incremental verification of the Layer 3 (L3) attributes and routing cannot be performed. There is a significant shortcoming in the existing real-time incremental verification of the configuration. In order to effectively avoid the configuration problems of the data link layer and the network layer in the future when the configuration is changed, we can solve the problem of the data plane to a certain extent by combining control plane verification and dynamic monitoring of configuration changes.

2.3 Recent New Verification Approaches and SDN Verification Work

Currently, some scholars have done verification work or network optimization for the verification of the control plane and data plane for traditional networks or SDN [20,21]. Among them, Chhabra et al. [22] validated the strategy in a federated cloud environment. Arunachalam et al. [23] constructed a mathematical model of the mp2p network and verified the search protocol in the network. Khalaf et al. [24] used the Deterministic PDA (DPDA) method to verify the web attack problem. Although he is not a verification method for the network, it is worth learning from to verify whether the network will be attacked.

3 Motivation

In this section, we present a toy example to show a common configuration update and conclude its affected area. Then we illustrate the opportunities for us to improve.

3.1 Example of Configuration Changes and Effect

As shown in Fig. 1, the left cloud represents Autonomous System 1 (AS1), and the right one represents AS2. In the following, we refer to them as domains and we omit other nodes in the network here. The left and right AS are composed of one core switch and two border switches. Open Shortest Path First (OSPF) is used as the routing protocol in each domain, and a BGP link is established between the two domains. AS1Border1 and AS2Border1 use BGP to connect the two domains, and AS1Border2 is connected to other network domains through ACCESS switches with BGP.

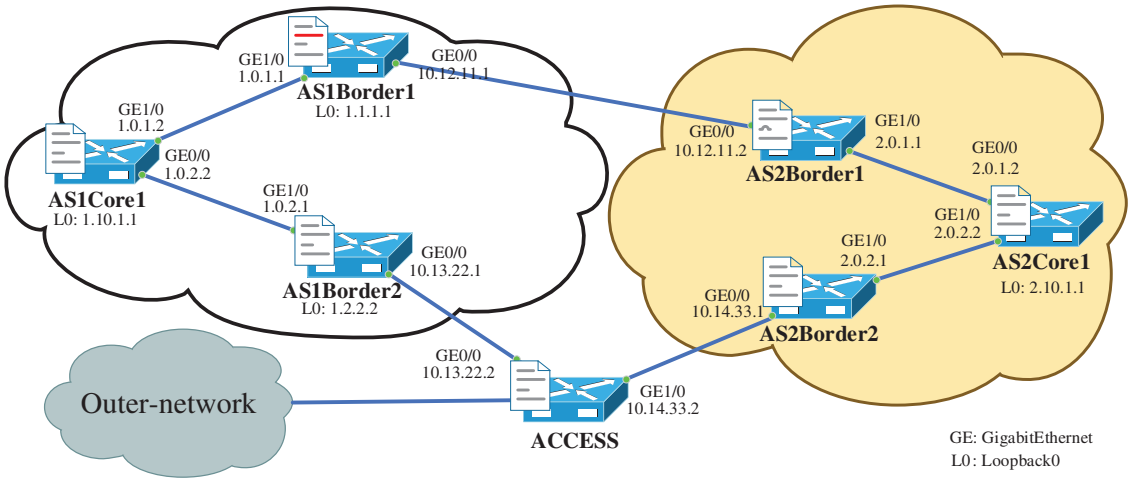


Figure 1: An example of configuration changes in a single line

Under normal circumstances, the two domains can communicate normally. At this time, for security or business adjustment considerations, the operator adds an ACL on AS1Border1 to isolate the specific flow (10.100.1.0/24) and make it impossible to pass AS1Border1 to reach AS2. After the network is changed, the network generally needs to be verified again to determine whether the configuration affects the security of the entire network. In this case, the affected network domain only involves a flow with a specific prefix, a path passed by the flow, and two domains.

Consider an extreme situation. If the operator changes all the network devices in AS1 in the initial situation, add n devices that belong to the same network segment as AS1Core1 in AS1 and connect them to AS1Core1. These devices are connected through the OSPF protocol, so there is no need to modify the configuration information of AS1Core1. In this case, the affected network domain only involves AS1.

3.2 Opportunities for Improvement

For Batfish, verifying reachability or other issues require reconstruction of the data plane snapshot. In the case of up to 3 links in the network, the verification time for 0-100 network devices is as high as 10^6 seconds. For ARC, the sum of the verification time for the four supported attributes reached 10^3 seconds in a network of 300 devices. Because Minesweeper is based on the SMT solution,

the verification time is well controlled, the average verification time for a small network of about 100 devices is about 10^1-10^2, and the average verification time for 400 devices is 10^3. For a single line configuration change, using Minesweeper's method to analyze the global configuration file line by line and the abstract modeling method will appear awkward and time-consuming.

In large-scale networks, most of the configuration changes are still only several lines or within one domain, one or two domains, as mentioned in the previous example. Global configuration verification is extremely time-consuming, and most of the time is spent in the process of verifying the correct configuration. The incremental verification and partial verification of the configuration solve this problem well.

The current incremental verification works mainly focus on the data plane, and we can solve the problem of the control plane through the data plane. Batfish can be used as an entry point and used to generate the data plane of the current network according to the configuration and provide the forwarding information table on each device. Through the forwarding information table, we can combine the data plane incremental verification tool to verify the network problems after the configuration change, but the results obtained after trying to use Netplumber and AP are not satisfactory. Therefore, it is possible to model the data plane, develop a verification tool for the data plane under the condition of determining the scope of influence of the configuration change, and complete the partial verification of the control plane in combination with Batfish.

4 Model

To better determine the impact area of the change in the network environment after the configuration change and carry out the next phase of verification work according to this area, we propose our flow-based model, in which we model the nodes in the network and their FIB based on flows, and we adopted a graph structure.

4.1 Flow Model

We define the concept of a *flow*, which includes the source device, source interface, source Internet Protocol (IP) address, destination IP address, and optional application and transport layer protocols. The label, definition, and type or range are listed below in Tab. 2. The Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) in the table are transport layer protocols. DNS, Secure Shell (SSH), and Simple Mail Transfer Protocol (SMTP) in the table are application layer protocols.

Table 2: Components of the flow model

Symbol	Definition	Type or Range
SrcDevice	The source location of the flow.	String
SrcInterface	The source interface of the flow in *SrcDevice*.	Loopback0, GigabitEthernet i/j, (i, j in (0,n)).
SrcIP	The source IP of the flow.	(0,0,0,0) – (255.255.255.255)
TProtocol	The transport layer protocol in which the flow is used.	{TCP, UDP}
DstIP	The destination IP of the flow.	(0,0,0,0) – (255.255.255.255)
application	The shorthand for application protocol which the flow used.	{DNS, SSH, SMTP, etc.}

4.2 Graph Model

After the determination of flow, we abstract the network into a graph structure, and each vertex represents a network device. Because each network device has a FIB table, as long as the flow is determined, the entries in the FIB table will indicate the remaining nodes that the flow can connect to after flowing through the vertex. This is similar to determining a point in the graph to find the minimum spanning tree. The cost required to find the minimum spanning tree can be set by the metric and Administrative Distance (AD) in the FIB table. The edges involved in each verification process are different, so the edge settings can be generated specifically according to different verification attributes. For example, in the example mentioned in Section 3.1, we want to send a flow from the Loopback0 interface of AS1Core1 to 2.10.1.1/32 to resolve DNS, so we get the flow defined below.

Flow1: {*SrcDevice* = AS1Core1, *SrcInterface* = Loopback0, *SrcIP* = 1.1.1.1, *TProtocol* = UDP, *DstIP* = 2.10.1.1/32, *application* = DNS}.

Therefore, starting from AS1Core1, we review the FIB entries that the flow complies with, and determine the possible flow direction level by level, and we can get the structure shown in Fig. 2. GE is the abbreviation of Gigabit Ethernet.

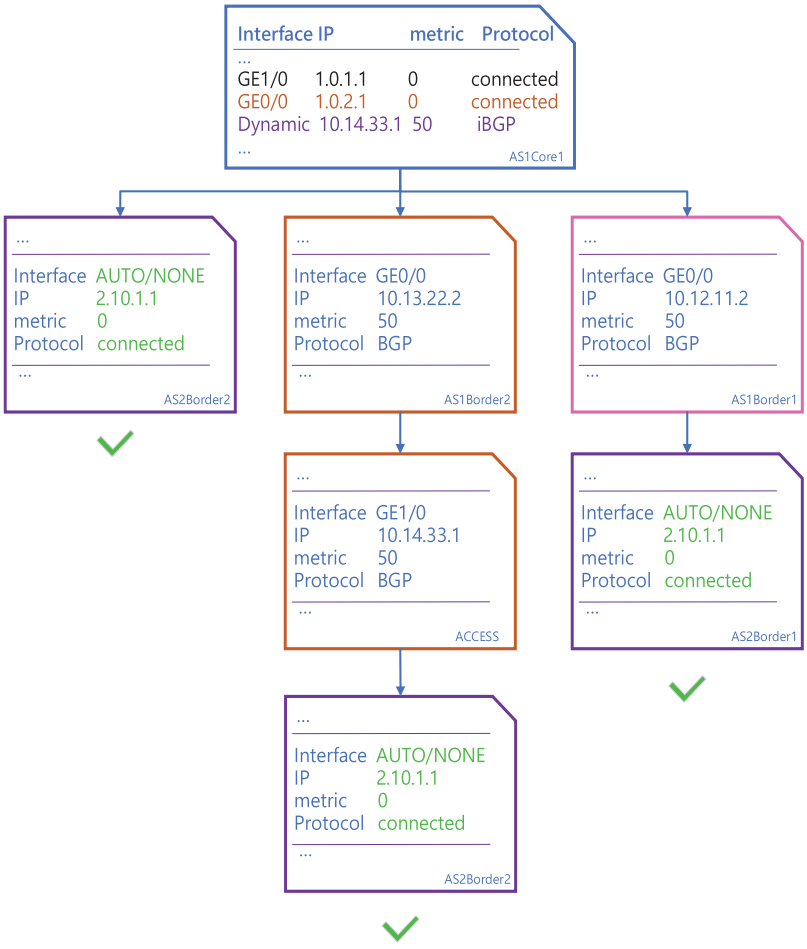


Figure 2: A sample graph of Flow1 in Fig. 1

5 Affected Area Determination

In this section, we classify and discuss the probable impact of configuration changes to FIB, and then define the network area that needs to be re-verified after changes.

5.1 FIB Table Entry Changes Caused by Different Configuration Changes

We compare the FIB tables before and after the entire network. At first, we classify the protocol of FIB entries, which are mainly divided into local, connected, OSPF, Internal/Interior BGP (iBGP), and BGP.

5.1.1 IP Changes

The change of the switch's IP will change the entries in the FIB table of this switch whose protocols are "connected" and "local". If the IP of the switch is modified, the network change of the entry whose next-hop interface type is Loopback0 will directly become the modified IP. If the switch interfaces' IP is modified, the network IP of the FIB entries with the next-hop interfaces type of "connection" will also be directly changed to the modified IP. At the same time, modifying the IP will change the FIB entries of the devices in the same OSPF domain and the FIB entries of devices that have established iBGP or BGP connection with the modified device. We divide the network devices (switches) whose IP changes have been made into core switches and border switches to discuss their effects.

For the core switch, whether the change occurs in the core switch can be discovered by checking whether the IP of the next device directly connected to the switch belongs to the same network segment as the IP of the switch. If it is in the same network segment, it means that the switch is a core switch. If the changed IP still belongs to the same OSPF advertised domain, then the direct connection interface network of the device directly connected to it will be modified. But if iBGP is configured on the device, the iBGP configuration will all be invalid. If the changed IP does not belong to the original OSPF domain, based on the previous impact, all entries belonging to OSPF in the FIB of the device will become invalid.

5.1.2 OSPF Changes

The change of OSPF involves the change of OSPF declaration. If only the new network segment is changed without modifying the advertised IP in an OSPF domain, then the OSPF protocol in this domain will be invalid, and the OSPF entries of all devices' FIB will be affected.

This change will also affect the BGP and iBGP protocols established between the boundary of the OSPF domain and other domains. That is, specific flows cannot be forwarded within an OSPF domain, and specific flows cannot be forwarded from border devices to other domains. If the device's IP is legally changed in an OSPF domain (legitimate here refers to the network segment that complies with OSPF declaration), OSPF will advertise the modified IP to the OSPF domain, and the change will be the same as the IP modification which is mentioned before. If an OSPF domain is not legally changed, the FIB entries of all devices in the OSPF domain whose protocol is OSPF will become invalid.

5.1.3 BGP and iBGP Changes

The change of BGP is more intricate. It involves the establishment and withdrawal of BGP and iBGP peers and the establishment and withdrawal of the route map. Since most network environments no longer use Routing Information Protocol (RIP) as Interior Gateway Protocol (IGP), we mainly

focus on BGP-based routing. The route map is predominantly used to realize the configuration of OSPF to RIP route redistribution and BGP-based route selection, and in the configuration, the route map is mainly used for local preference setting, metric setting, and ACL setting. It will affect the route chosen by BGP, the FIB entries that flow through the modified route-map configuration device whose protocol is BGP. The establishment or revocation of BGP or iBGP peers affects both the establishment of BGP or iBGP and the FIB entries in BGP or iBGP devices in the entire network domain.

5.1.4 ACL Changes

The change of an ACL for a specific flow forwarding policy will affect the FIB changes of all devices through which the flow may traverse. For some unavailable ACLs (such as an ACL entry for 3.0.0.0/8 IP in a network that only contains 1.0.0.0/8 and 2.0.0.0/8), their changes should not change the global FIB table. So, there is no need to define the scope of the impact of this change. The changed global device will take a detour method to establish a new FIB for a specific flow. The verification of the correctness of the ACL change needs to involve whether the user's intention is fully realized and where the specific traffic flow is successfully permitted or denied. This involves two points, one is whether the ACL is filtering the flow correctly in the correct position, and the other is whether the ACL does not affect the irrelevant flow. Inputting certain points, certain flows, and operations on certain flows can check the implementation status of ACLs, thereby solving these two problems. Intra-domain traffic will be resolved within the domain, and cross-domain traffic needs to track the set of domains that the traffic can reach.

5.2 Determine the Re-verification Range According to the Verified Attributes

5.2.1 Reachability

Reachability is proposed based on the flow. That is, whether a particular flow can reach another point in the network and it can be checked directly by observing the changes of global FIB. We can query whether the FIB of the network device where the flow is injected has a FIB entry that reaches the destination IP to check it.

However, in a large-scale network, the query time required for multiple reachability verifications is still very time-consuming, so the destination network address of the query can be compared with the FIB difference set before and after the change. If the destination network is not in the difference set, its reachability will not be changed, but if the destination network is in the difference set, the reachability between the two points may change. We take advantage of the complexity of graph algorithms by constructing a path from the source to the destination device to check this attribute, which eliminates unnecessary table lookups and considerably reduces the time required for verification.

5.2.2 Always Blocked

For security reasons, the network administrator may want a certain flow to be always blocked by one device. Therefore, to verify whether the attribute is violated after the change, we need to check whether there have relevant FIB entry changes in this device. If the device's new FIB has a network that contains flows that the administrator wants to not have, then this property may be violated. For more precise judgment, we need to track the certain flow to figure out whether it can reach the device that needs to meet always blocked.

5.2.3 Loop Detection

At first, we take the network before the change as a loop-free network. For the changed network, we have omitted the loop lookup for iBGP changes, because iBGP changes will not be notified to peers, and routing loops will not occur. Afterward, we need to look at the changed routes before and after the change. We will look for the changed routes in turn, record their path, and if the same network appears in the path, it is determined that the route has a loop.

5.2.4 Traverse Waypoint

The network administrator may want the flow that flows through a domain on the boundary to pass through a firewall, a waypoint. This is principally to verify whether any flow has reached the destination domain by detour. If the FIB difference caused by the network change does not involve the network of the domain, it does not need to be verified to show that it has not been violated. If the network of this domain is involved, it is necessary to inject a flow to track the device that it passes through. If the flow never flows through the device, it means that the attribute has been violated.

5.2.5 Traffic Isolation

For security reasons, the administrator may want to make the paths of two flows never cross one or a group of devices. We can check whether the FIB difference set before and after the configuration modification involves the destination addresses of the two flows. If not, no verification is required, and the attribute is still satisfied. It is required to record the paths that the two flows traverse when the destination address is involved and if the same device is included in the paths of the two streams, this attribute is violated.

5.2.6 Correct Implementation of ACLs and Route-map

The verification of this attribute is to prove whether the newly set ACLs and route map have been implemented correctly. After entering the point where the ACLs and route-map are set and obtaining the IP of the specific operation, we can check the FIB at that point to see if there is a next-hop network for a specific flow to get verification.

5.2.7 Summary

The verification of all the above attributes does not require re-verification of the entire network but only needs to adopt an additional method to determine the interval for verification, which reduces the time spent in verification. And the time consumption of configuration update verification is not expanded by the expansion of the network, so the verification efficiency will be enhanced in medium and large networks.

6 FastCUV

We propose FastCUV, a tool for fast configuration update verification. This section will introduce specific implementation algorithms of FastCUV. The initial FIB table we use is error-free and conforms to the deployment intent of the manager. The deployment intent here refers to whether the interaction between ASs has been defined, ACL configuration is correct, etc.

6.1 Flow Traceroute

The verification of many attributes in Section 5.2 requires recording the network devices through which the flow passes. Therefore, we present the traceroute algorithm shown in Fig. 3 as Algorithm 1.

Algorithm 1: Flow Traceroute

Input: a certain flow $Flow$,
Start point A,
End point B.
Output: $trace$: a list of nodes which the flow traverses,
$Flag$: a boolean type variable to show the reachability.

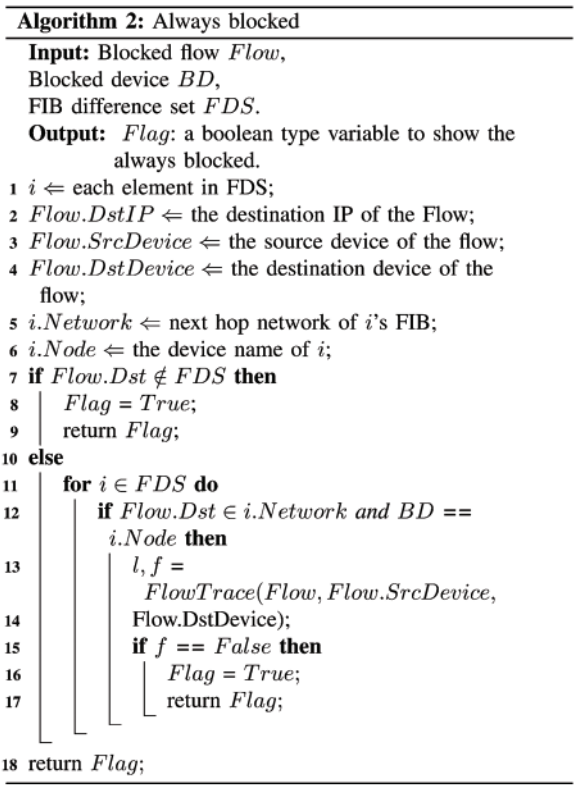
1 $A.Node \Leftarrow$ the device name of A;
2 $A.FIB \Leftarrow$ the FIB belongs to A;
3 $Flow.DstIP \Leftarrow$ the destination IP of the flow;
4 $B.IP \Leftarrow$ the IPs belongs to B including Loopback0 and interfaces;
5 $i.Netowrk \Leftarrow$ next hop network of i's FIB;
6 $i.Node \Leftarrow$ the device name of i;
7 **for** i *in* $A.FIB$ **do**
8 **if** $Flow.DstIP = B.IP$ **then**
9 $trace.append(i.Node)$;
10 $Flag$ = $True$;
11 return $Flag.trace$;
12 **else if** $Flow.DstIP \notin i.Network$ **then**
13 $Flag$ = $False$;
14 return $Flag, trace$;
15 **else**
16 A = $i.Node$;
17 $trace.append(i.Node)$

Figure 3: Algorithm1: Flow trace route

The input of the algorithm is *Flow*, which is the specific flow to be traced, the source point A and the destination point B of the flow. The output is *trace*, a list of traces flowing through points, and a Boolean *Flag* to determine whether the two points are reachable. Lines 1–6 of the algorithm describe the required variables. Line 7 adds the source node to the *trace*. Lines 8–17 enter the loop. Lines 9–12 determine whether the current node is directly connected to the destination node. If so, add the current node to the *trace*, and then return *Flag* with a true. Lines 13–15 indicate that if the destination node of *Flow* does not belong to the FIB table of the current node, the network is unreachable, and returns *Flag* with a false. Line 16 indicates that if the destination address of *Flow* is in the FIB of the current node, then let this node be the point to continue the loop and add this node to the trace. Similarly, the algorithm can also be used when the destination network is located in the FIB difference set when verifying reachability. We name this algorithm *FlowTrace( )* in the algorithms below.

6.2 Always Blocked

The verification algorithm for always blocked is shown in Fig. 4 as Algorithm2. The input of the algorithm is the *Flow* that operators want to block, the device *BD* for which the policy takes effect, and the FIB difference set *FDS* of the FIB table before and after the change. The output is a Boolean *flag* to indicate whether the verification attribute is violated.



Algorithm 2: Always blocked

Input: Blocked flow $Flow$,
Blocked device BD,
FIB difference set FDS.
Output: $Flag$: a boolean type variable to show the always blocked.

1 $i \Leftarrow$ each element in FDS;
2 $Flow.DstIP \Leftarrow$ the destination IP of the Flow;
3 $Flow.SrcDevice \Leftarrow$ the source device of the flow;
4 $Flow.DstDevice \Leftarrow$ the destination device of the flow;
5 $i.Network \Leftarrow$ next hop network of i's FIB;
6 $i.Node \Leftarrow$ the device name of i;
7 **if** $Flow.Dst \notin FDS$ **then**
8 $Flag = True$;
9 return $Flag$;
10 **else**
11 **for** $i \in FDS$ **do**
12 **if** $Flow.Dst \in i.Network$ *and* $BD == i.Node$ **then**
13 $l, f = FlowTrace(Flow, Flow.SrcDevice,$
14 Flow.DstDevice);
15 **if** $f == False$ **then**
16 $Flag = True$;
17 return $Flag$;
18 return $Flag$;

Figure 4: Algorithm2: Always blocked

Lines 1-7 describe the variables that appear in the algorithm. Line 8 starts to judge, and if there is no modification of *Dst* related to *Flow* in *FDS*, then the *Flag* with the return value of true from the exit and judgment is proved to be violated. In line 11, if the previous conditions are not met, enter the else, here enter the loop, analyze each entry of the *FDS*, determine the entry related to the *Dst* of *Flow*, and the changed device is the device deployed by always blocked, use *FlowTrace( )* to track the flow. If it is not reachable, it means that always blocked is still satisfied, and the return value is true, otherwise, this attribute will be violated and the return value will be false.

6.3 Loop Detection

The loop detection algorithm is shown as Algorithm 3 in Fig. 5. The input of this algorithm is the FIB difference set *FDS* that eliminates iBGP. The output is a Boolean *Flag* to indicate whether there is a routing loop and a list *Llist* that contains loops. Lines 1-9 of the algorithm describe the variables that appear in the algorithm. Line 10 enters the loop, traverses all the entries in the DFS, generates a flow for each entry, and gets the device through which the flow traverses through *FlowTrace( )*. Then line 13 enters the second layer loop and analyzes these devices. If these devices have only appeared once, then it can be determined that there is no routing loop and the algorithm continues. If there are some devices that appear more than once, then the FIB is added to the *Llist*. Start the judgment from line 18, if *Llist* is empty, then there is no loop, otherwise, it means that there is a routing loop.

Algorithm 3: Loop Detection

Input: FIB difference set FDS;
Output: $Flag$: a boolean type variable to show the existence of loop;
$Llist$: a list contains routes which have forwarding loops;

```
1  Flow ⇐ a flow generated by FDS;
2  i ⇐ each element in FDS;
3  Flow.DstIP ⇐ the destination IP of the Flow;
4  Flow.SrcDevice ⇐ the source device of the flow;
5  Flow.DstDevice ⇐ the destination device of the
     flow;
6  i.Network ⇐ next hop network of i's FIB;
7  i.Node ⇐ the device name of i;
8  l ⇐ Node list returned by FlowTrace();
9  f ⇐ Boolean flag returned by FlowTrace()
10 for i in FDS do
11     Flow = (SrcDevice = i.Node,DstIP =
         i.Network;
12     l, f = FlowTrace(Flow, Flow.SrcDevice);
13     for j in l do
14         if j not equal ∀k ∈ l then
15             pass;
16         else
17             Llist.append(i)
18 if Llist.Empty() then
19     Flag = True;
20     return Flag;
21 else
22     Flag = False;
23     return Flag;
```

Figure 5: Algorithm3: Loop detection

6.4 Traffic Isolation

The Traffic Isolation algorithm is shown as Algorithm 4 in Fig. 6. The input of the Traffic Isolation algorithm is the FIB difference set *FDS*, the point *Mark*, and two Flows *Flow1* and *Flow2* that the operator does not want to pass through a single point at a different time. The output of the algorithm is a Boolean *Flag* to indicate whether traffic isolation is violated. Lines 1–4 describe the variables appearing in the algorithm. Lines 5 and 6 use *Flowtrace( )* to get the list of nodes *l1* and *l2* passed by the two flows. Lines 7–10 are used to judge if the *Mark* points do not exist in the two lists at the same time, then return false, otherwise, return true.

Algorithm 4: Traffic Isolation

Input: FIB difference set FDS;
an isolation point $Mark$;
two certain Flow $Flow1, Flow2$;
Output: $Flag$: a Boolean type to show the Traffic Isolation;
$Flag$: a boolean type variable to show the reachability.

```
l1 ⇐ Node list which the Flow1 traverses returned by FlowTrace();
f1 ⇐ Boolean flag of Flow1's reachability returned by FlowTrace();
l2 ⇐ Node list which the Flow2 traverses returned by FlowTrace();
f2 ⇐ Boolean flag of Flow1's reachability returned by FlowTrace();
l1, f1 ⇐ FlowTrace(Flow1);
l2, f2 ⇐ FlowTrace(Flow2);
if Mark ∈ l1 and Mark ∈ l2 then
    Flag = False;
    return Flag;
else
    Flag = True;
return Flag
```

Figure 6: Algorithm4: Traffic isolation

7 Evaluation and Analysis

We use two virtual machines running Ubuntu20.04 as Batfish servers, enter the modified configuration, and the modified configuration into the two servers, respectively, and use a Windows PC as a client to interact with it.

We have injected some bugs into the configuration file. These bugs are common faults, such as user input errors, IP address changes, OSPF configuration changes, BGP configuration changes, etc. According to section 2, we chose Batfish and ARC, the typical open-source control plane verification tools, to compare our work. The results of the comparison are shown in Tab. 3.

We compared them by the ability to verify common faults and the ability to support some attributes compare them with faults verification ability and some attributes support. The symbols + and – denote that the flaw, bug or attribute can and cannot be detected or supported by the verifier, respectively.

Table 3: Verification results of configuration errors

No.	Faults and attributes	Batfish	ARC	Ours
1	Always blocked	–	+	+
2	Always reachable with < k failures	+	+	–
3	Always isolated	–	+	+
4	Always traverse waypoint	–	+	+
5	Data plane and control plane equivalent test	–	+	–
6	Manual input configuration error detection	+	+	+

(Continued)

Table 3: Continued

No.	Faults and attributes	Batfish	ARC	Ours
7	Configuration update verification support	–	–	+
8	Forwarding Loop detection	+	+	+
9	BGP and iBGP configuration verification support	+	–	+
10	Blackhole	NA	+	+
11	Multipath consistency	+	–	–

We tested six different types of small local area networks, with equipment ranging from 10 to 30. Verified for Always Blocked, Loop Detection, Traverse Waypoint, Traffic Isolation, Correct implementation of ACLs and route-map. The configuration changes vary from one to 100 and the experimental results are shown in Figs. 7 and 8.

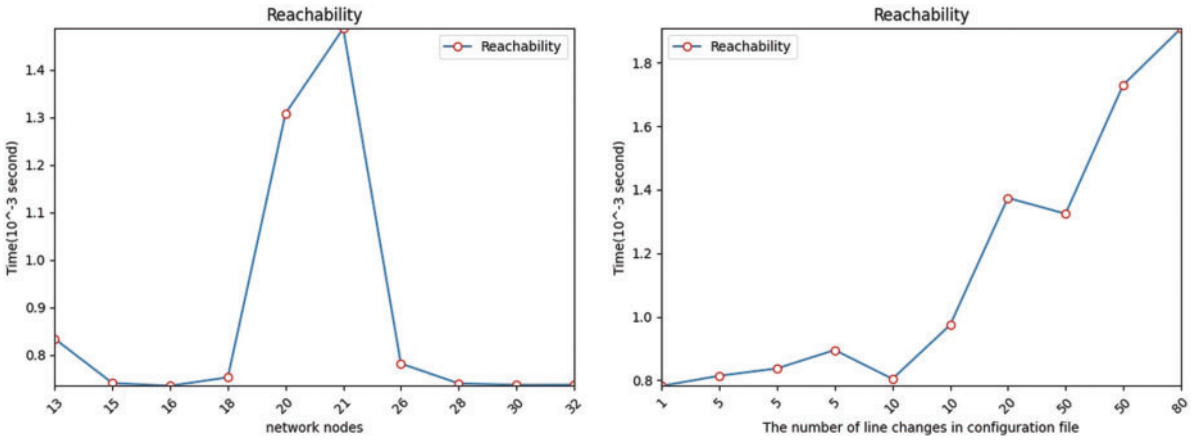


Figure 7: Verification time of reachability

It can be seen that for small-scale networks and less than 100 lines of configuration changes, the verification speed of Reachability is 10^{-2}–10^{-3} seconds, and the verification speed of Loop Detection is 10^{-1} second. The increase in the number of network nodes does not correlate with the verification time of the two, but the number of changed lines in the configuration file is positively correlated with the time spent in verification.

8 Future Work and Discussion

In this paper, the fast control plane verification is carried out based on the configuration changes of the traditional network distributed control plane. We propose a flow model to better judge the scope of influence of configuration changes and to facilitate subsequent verification work. We discuss the possible effect of configuration changes on the FIB table, and the verification area is determined according to the different attributes that need to be verified. And we propose and implement FastCUV, a verifier for verification of fast configuration updates. Experiments have proved that the expansion of the network does not affect the efficiency of our work, but the number of rows of configuration changes has a greater impact.

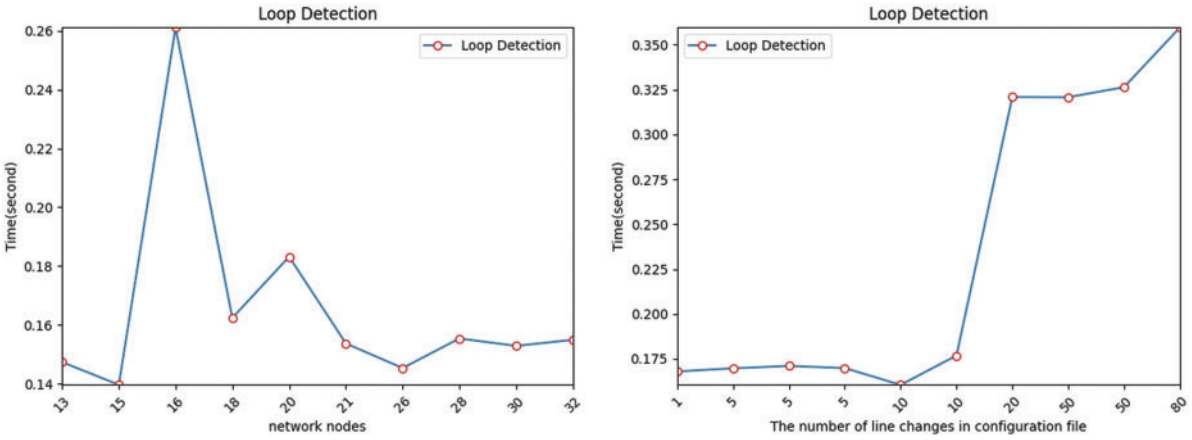


Figure 8: Verification time of loop detection

This paper does not explain other attributes due to space limitations and will be completed in the follow-up work. Incremental verification for k-failures, which has the greatest impact on global verification, is under development and has not yet been completed, and will be completed in the next step. The conclusions still need to be verified in a large-scale network, and then the configuration files of the large-scale network will be screened for experimental analysis.

Acknowledgement: We are thankful to all the collaborating partners.

Funding Statement: This work was supported by the Fundamental Research Funds for the Central Universities (2021RC239), the Postdoctoral Science Foundation of China (2021 M690338), the Hainan Provincial Natural Science Foundation of China (620RC562, 2019RC096, 620RC560), the Scientific Research Setup Fund of Hainan University (KYQD(ZR)1877), the Program of Hainan Association for Science and Technology Plans to Youth R&D Innovation (QCXM201910) and the National Natural Science Foundation of China (61802092, 62162021).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] T. Benson, A. Akella and A. Shaikh, "Demystifying configuration challenges and trade-offs in network-based isp services," in *Proc. of the ACM SIGCOMM, 2011 Conf.*, Toronto, Ontario, Canada, pp. 302–313, 2011.

[2] H. Kim, T. Benson, A. Akella and N. Feamster, "The evolution of network configuration: A tale of two campuses," in *Proc. of the 2011 ACM SIGCOMM conf. on Internet measurement conf.*, Berlin, Germany, pp. 499–514, 2011.

[3] Y. Sung, X. Tie, S. H. Y. Wong and H. Zeng, "Robotron: Top-down network management at facebook scale," in *Proc. of the 2016 ACM SIGCOMM Conf.*, Florianopolis, Brazil, pp. 426–439, 2016.

[4] A. Gember-Jacobson, W. Wu, X. Li, A. Akella and R. Mahajan, "Management plane analytics," in *Proc. of the 2015 Internet Measurement Conf.*, New York, NY, USA, pp. 395–408, 2015.

[5] Y. Sung, S. Rao, S. Sen and S. Leggett, "Extracting network-wide correlated changes from longitudinal configuration data," in *Int. Conf. on Passive and Active Network Measurement*, Seoul, Korea, pp. 111–121, 2009.

[6] S. Vissicchio, L. Vanbever, C. Pelsser, L. Cittadini, P. Francois *et al.,* "Improving network agility with seamless BGP reconfigurations," *IEEE/ACM Transactions on Networking*, vol. 21, no. 3, pp. 990–1002, 2012.

[7] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan *et al.,* "A general approach to network configuration analysis," in *12th USENIX Sym. on Networked Systems Design and Implementation (NSDI 15)*, Santa Clara, CA, USA, pp. 469–483, 2015.

[8] J. Yao, Z. Jiang, W. Yang, M. Wang and D. Li, "Forwarding Policy Verification of Hybrid Multi-domain Software-Defined Network," in *IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. Haikou, China, 2371–2378, 2021.

[9] H. Mai, Haohui, A. Khurshid, R. Agarwal, M. Caesar *et al.,* "Debugging the data plane with anteater," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 290–301, 2011.

[10] P. Kazemian, G. Varghese and N. McKeown, "Header space analysis: Static checking for networks," in *9th USENIX Sym. on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, USA, pp. 113–126, 2012.

[11] A. Khurshid, X. Zou, W. Zhou, M. Caesar and P. B. Godfrey, "{VeriFlow}: Verifying {Network-Wide} Invariants in Real Time," in *10th USENIX Sym. on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL, USA, pp. 15–27, 2013.

[12] H. Yang and S. S. Lam, "Real-time verification of network properties using atomic predicates," *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 887–900, 2015.

[13] A. Gember-Jacobson, R. Viswanathan, A. Akella and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proc. of the 2016 ACM SIGCOMM Conf.*, New York, NY, USA, pp. 300–313, 2016.

[14] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson and Aditya Akella, "Tiramisu: Fast multilayer network verification," in *17th USENIX Sym. on Networked Systems Design and Implementation (NSDI 20)*, Santa Clara, CA, USA, pp. 201–219, 2020.

[15] R. Beckett, A. Gupta, R. Mahajan and D. Walker, "A general approach to network configuration verification," in *Proc. of the Conf. of the ACM Special Interest Group on Data Communication*, Los Angeles, CA, United States, pp. 155–168, 2017.

[16] S. Son, S. Shin, V. Yegneswaran, P. Porras and G. Gu, "Model checking invariant security properties in OpenFlow," in *2013 IEEE Int. Conf. on Communications (ICC)*, Budapest, Hungary, IEEE, pp. 1974–1979, 2013.

[17] S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *17th USENIX Sym. on Networked Systems Design and Implementation (NSDI 20)*, Santa Clara, CA, USA, pp. 953–967, 2020.

[18] E. AI-Shaer and S. AI-Haj, "FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures," in *Proc. of the 3rd ACM workshop on Assurable and usable security configuration*, New York, NY, USA, pp. 37–44, 2010.

[19] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown *et al.,* "Real time network policy checking using header space analysis," in *10th USENIX Sym. on Networked Systems Design and Implementation (NSDI 13)*, Lombard, IL, USA, pp. 99–111, 2013.

[20] R. Malavika and M. L. Valarmathi, "Adaptive server load balancing in sdn using pid neural network controller," *Computer Systems Science and Engineering*, vol. 42, no. 1, pp. 229–243, 2022.

[21] G. R. Sreekanth, S. Ahmed, M. Sarac, I. Strumberger, N. Bacanin *et al.,* "Mobile fog computing by using sdn/nfv on 5g edge nodes," *Computer Systems Science and Engineering*, vol. 41, no. 2, pp. 751–765, 2022.

[22] N. Chhabra, M. Bala and V. Sharma, "Implementation and validation of the optimized deduplication strategy in federated cloud environment," *Computers, Materials & Continua*, vol. 71, no. 1, pp. 2019–2035, 2022.
[23] A. Arunachalam, V. Ravi, M. Krichen, R. Alroobaea and S. Rubaiee, "Mathematical model validation of search protocols in mp2p networks," *Computers, Materials & Continua*, vol. 68, no. 2, pp. 1807–1829, 2021.
[24] O. I. Khalaf, M. Sokiyna, Y. Alotaibi, A. Alsufyani and S. Alghamdi, "Web attack detection using the input validation method: dpda theory," *Computers, Materials & Continua*, vol. 68, no. 3, pp. 3167–3184, 2021.