NCF: A secure software defined networking framework to avoid vulnerabilities

Sandeep Surendher
R.M.D Engineering College
Anna University
umailsandy@gmail.com

Shweta Sridharan
R.M.D Engineering College
Anna University
shwetasridharan@gmail.com

Abstract—The software defined networking dissevers the control plane from the forwarding elements. The controller in the control plane has holistic network visibility and programmability thus bringing in greater innovations in networking as such. A concentrated view of the network is provided to the applications like traffic signaling, load balancing etc. while contemporarily increasing the security and dependability. Everything has its own cons and so does the SDN. At some level, the security of SDN is an issue. There are many SDN specific threats such as data to control plane saturation attack, host location hijacking which questions its deployment in critical networks.
These occur primarily because the southbound interface is currently standardized by an open flow framework which is simple and vulnerable to threats. This paper addresses the above issue and aims to create a secure (bulwark) SDN framework wherein an additional layer of hierarchy is provided in the existing network. Inspired by the programmable networks (PN), we introduce a node based controlling and forwarding (NCF) framework in the SDN network. We offload some of the intelligence from the centralized control plane and divide the distributed data plane into domains.
The nodes establish a secure connection with the controller. They maintain the flow table for all the switches connected to them. Hence an attacker can no longer exploit the vulnerabilities of the TCAM memory of the switches to invoke fake flows. This paper discusses all the issues in the current SDN using open flow and how they can be rectified using a more secure framework.

Keywords—Software-Defined Networking (SDN); Network Architecture; Scalability; Denial-of-Service attack;

INTRODUCTION

SDN using open flow directs each micro flow to a centralized controller that reactively installs the appropriate flow rules in the switches. Our node based framework proposes to proactively install flow rules by dividing the network into distributed domains. Based on the network topology changes reported to the controller, these flow rules can be changed as and when it is needed. This prevents the delay time occurring while processing new flows. We also divide the intelligence among the network nodes thereby reducing the burden on the controller.
Our framework proposes to provide enhanced security and scalability by overcoming many of the challenges present in the open flow. This paper is divided into the following sections:
Section 1: provides a comprehensive review of the issues present in the current SDN.
Section 2: provides a bulwark SDN framework which can be scaled and updated easily.
Section 3: shows how our framework and the algorithm we have used solves the issues we discuss in the section 1.

I. ISSUES

The current model of SDN installs the flow rules on a coarse basis, that is provides granular control thus empowering the network to deal with real time challenges at the application, user and session level. The very first packet of each micro flow is directed to a centralized controller that relatively installs befitting flow rules in the switch.

A. Signalling overhead delay in reactive approach

If a table-miss occurs in the switch due to unknown flows, then the controller must be contacted. The packet header is forwarded to the controller, meanwhile, keeping the packet in the switch's buffer memory.
The controller after prospecting the packet header modifies the flow table of the switch using the flow-mod message and thus the packet is forwarded. This leads to a substantial delay and increases the response time for handling new flows.

B. Switch complexity

A flow is primarily a collection of similar packets. For each new flow, packets are initially kept in the buffer memory (especially during reactive flow). On the event of biggish flows, the buffer memory of the switch will not suffice for

storing the packets. This problem is more prevalent if the packets are from UDP agents. The reason being, multiple packets of the same flow can be in the flight. Hence, each switch needs to maintain a complex buffer mechanism in order to preserve the packet during the response time from the controller.

C. *Misbehaving hosts*

A malicious user can easily exploit the vulnerabilities in the switch's TCAM memory and forward fake flows to create an overflow and thereby create transient loops. Multitudinous table-miss packets may cause the switches' buffer memory to go full. As a result of all this, the switch goes down.

We might think that, using proactive flow rules can curb this problem. But TCAM (Ternary Content Addressable Memory) is limited, that is even if flow rules are installed proactively in all the switches, it is not feasible for the switches buffer memory) to store all the flow rules with the relatively small TCAM memory in the switches. This in turn augments the complexity of switches when the network needs to be scaled.

Another setback we face even after opting for proactive flow is- no mobility support. The proactive flow rules downgrades the dynamic capability of switches to process the packets. To elaborate, low level flow rules when installed in advance, do not scale well in mobile hosts. Since same flow rules need to be installed each time in multiple locations as and when the host moves from one switch to another. It also gives rise to problems in prioritizing. Some traffic from different departments of the same hosts need to be treated otherwise.

D. *Link failures*

It is an uphill task for an SDN using open flow to recover from a link failure within say 50ms (typical requirement for voice traffic).

Network devices loose connectivity with the controller if the primary uplink fails. Offloading some of the intelligence and installing pre-computed alternate paths into the network nodes is one solution. In simpler words, intelligence must be intricate instead of routing it centrally. This reduces the overload on the controller and provides a leg-up in scaling networks. For this, the data plane must also be node-programmable.

Google recognized these challenges and hence uses open flow in their G-scale networks, only within a data center, where a cluster of open flow controllers manage local devices. They use traditional routing protocols between sites and further manage traffic flows with proprietary TE similar to MPLS.

Updating issue connected with Open flow:

E. *Updating issue*

If a new application like say load balancing is to be introduced in the network, whose operations the current open flow does not support, then it needs to be updated. The control plane needs to update all the switches in the network to support the required operations. Individually updating all the switches may be error prone and lead to mismanaged updating. It will be time consuming too. For example, routing loops. Therefore we need a programmable data plane to control a particular set of switches which when updated, updates all the switches.

F. *Data to control plane saturation attack*

In Open Flow switches, When a table-miss occurs, that is there is a new packet which data plane does not know how to handle, the data plane will buffer the packet and send a packet_in message which contains the packet header to controller if the buffer memory is not full. If the buffer is full, the message will contain the whole packet instead of only its header. The attacker can exploit it by launching dedicated data-to-control plane saturation attack that floods SDN networks. He can generate a large number of anomalous packets, which means all or part of fields of each packet are spoofed as random values. These spoofed packets have a low probability to be matched by any existing flow entries in the switch. As a result, the flooding attack will significantly downgrade the performance of the whole Open Flow infrastructure.

In simpler words, data to control plane saturation attack first exploits the vulnerabilities in the switch to flood its buffer memory .The control plane is then flooded with packets causing the brain of the network to fail.

Many papers have proposed to solve this issue. For example, Floodgaurd installs proactive flow rules to separate benign packets from the malicious ones. This makes the whole data plane to go into a state of defense limiting the functionality of the data plane. In a busy network this state of defense may in turn increase the load of control plane and cause the flooding of control plane, apart from the attack caused by saturation. Although the flood guard proposes to update the proactive flow rules, it is very difficult to handle the rules by competing with the speed of attacker. Hence we need to divide the network into domains to make most part of the network to work during an attack.

G. *Network topology and host location tracking*

A wide visibility of the entire network is one of the key coign of vantage provided by the SDN compared to the conventional networking technologies. The controller and the upper application layer harness this topology information.

However, if such fundamental and rudimentary network topology information is poisoned, all dependent network services will get affected. Security and usability becomes an issue. In datacenter, it is drudging and error prone to maintain the location of virtual switches due to their frequent migration. In this regard, HTS (Host Tracking Service) in the controller is used to provide a facile way to generate flexible network dynamics by dynamically probing packet_IN and updating host profile. DPID of the switch on ingress port ID is noted and by monitoring packet-IN messages, we grasp that the host has moved.

Upon the exploitation of the Host Tracking Service, an attacker can hijack the location of a network server to phish its service subscribers. By poisoning the Link Discovery Service, an adversary can inject false links to create a black-hole route

or launch a man-in-the-middle attack to eavesdrop/manipulate messages in the network.

Here, we propose an attacking strategy where the adversary crafts packets with the same identifier of the target host. Once receiving the spoofed packet, the Open Flow controller will be tricked to believe that the target host moves to a new location, which actually is the attacker's location.

II. ARCHITECTURE.

We now discuss the architecture of the proposed node based framework.

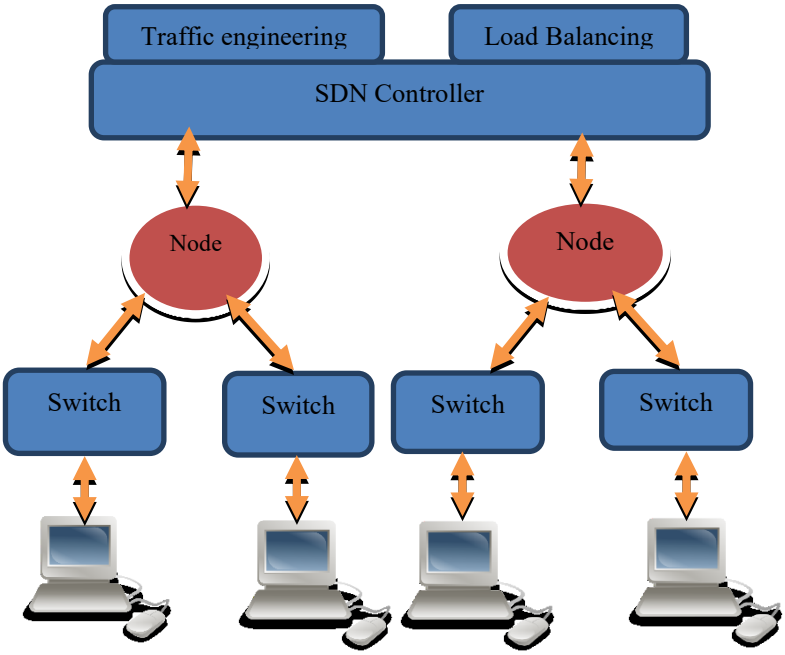
A. *Abbreviations and Acronyms*

NCF, DVR, TE, MLPS, TCAM

B. *Framework*

In the present SDN framework, the data forwarding layer is located at the bottom of the SDN architecture. It contains thousands of switches interconnected with each other. These switches maintain the flow table for forwarding the packets. If a switch is compromised or tampered with, the packets that flow through it will be forwarded incorrectly.

But in our NCF (Node Based Control and Forwarding) framework, the switches at the bottom layer of the hierarchy do not maintain the flow tables. Instead, the nodes maintain the flow table for all switches that are directly connected to it .Thus a large data centre can be divided into domains and each domain can be maintained by a particular host of switches and a node for all those switches.



C. *Host tracking*

Switches can track their directly connected hosts by monitoring the MAC address table stored in it. This MAC address table provides us with the information of the hosts along with their respective port numbers. For instance, consider the frame below:

00:0a:95:9d:68:16	Fa 0/1

The switches face two major situations:

- Whenever a new host or a user gets added to a switch, the switch tracks the new user and updates the Mac table.
- Whenever a host changes from one port location to another in the same switch, discordant MAC entries will be present. The switch thus removes the old MAC entry and replaces it with the new MAC entry.

Switches can also track link failure or host failure. The nodes in general, maintain a host profile for all the directly connected switches. In that context, whenever a new switch connects to the node, the node assigns a switch ID. Thus the previous frame contains another block of switch ID.

23	*00:0a:95:9d:68:16*	Fa 0/1

The switches keep informing the node about any updates in the MAC table. The nodes in turn make use of this information from all the switches to maintain the individual host profiles. The host profile contains the following requisites: MAC address, IP address and the port number. The host profile is generally indexed by the MAC address.

D. *Updating host and link failures*

The hosts should send a "HELLO" message periodically (periodic updates-say 5 seconds) to prove its active state. If the periodic update goes missing for even two consecutive periodic cycles, the host is considered to have migrated to some other location or the link/host has failed. Either way, the MAC entry for that particular host is removed and the node in turn gets updated by removing that host's profile.

This dynamic and glitch free updating is what makes it more beneficial and favorable as compared to the traditional network. In the data center, it is very tedious to maintain the location of virtual machines due to their frequent migration. Host tracking service provides a way to assure flexible network dynamics by constantly updating host profile in the node. In general, the controller must be aware of the host location changes and link failures to globally maintain the host profiles. This in turn helps the controller to perform traffic engineering, load balancing and other applications.

E. *Calculating network topology*

The nodes which maintain the host profiles of their domain need to advertise their information to a centralized controller which maps the entire network. This can be done using many algorithms in networking. We will use the Distance Vector Routing (DVR). The reason being, it is a very light weight, simple and low cost algorithm. Traditionally according to the

algorithm, the nodes inform their respective directly connected neighbors about the topology changes periodically. But in our algorithm, the nodes advertise to the controller, the distance and direction to each of its directly connected switches. Collectively, the controller uses all this information to calculate the network topology.

Every node connected to the controller is assigned a data path ID (DPID) by the controller itself. Now the controller can identify the nodes based on their DPID and port number.

Thus, updates take place periodically using DVR algorithm in order to inform the controller about any switch changes or host profile changes. Thus this process reduces the delay time involved in reactively installing flow rules by using open flow.

Now the question is, how it actually works. The controller merges all the information advertised by the nodes to create a network topology which can be reactively updated based on the changes in the network. After mapping the entire topology, an abstraction of the network is provided to the north bound interface. The application layer makes use of this information to perform tasks like traffic engineering, load balancing etc. A global host profile maintained by the controller will be updated dynamically based on the host changes which are updated by local host profiles in nodes. For instance, imagine there is a host profile change advertised by node. The controller makes use of the DPID of the node and the respective switch ID to instantly match it with the global host profile. When a difference is detected, the old entry is replaced with the new entry, thereby updating the host profile.

44	23	*00:0a:95:9d:68:16*	Fa 0/1
DPID	**switch ID**	**port number**	**MAC add.**

Controller faces the challenges of merging and leveraging the entire data produced by the network. The DPID and the switch-ID are quintessential elements in matching and grouping the nodes for partitioning rules and identifying individual hosts.

F. Partitioning and distributing flows

Once the network topology has been calculated, the flow rules must be forwarded proactively to their respective nodes. This adversely helps in reducing the delay and the response time which otherwise will be more, given that each new flow will need to contact the control plane individually. Another advantage is that, the local traffic within the node stays in the data plane due to proactive forwarding of the flow rules.

Partitioning of flow rules is relatively simpler in this node based framework than other similar approaches you would have come across. For example, DIFANE uses authority switches to flood proactive rules. In our network topology, nodes receive proactive flow rules only for the set of switches directly connected to it.

Thus we offload the intelligence from the centralized control plane to enable faster look ups and thereby reducing the delay time involved in processing new flows.

G. Flow handling

When a new flow reaches the switch, it checks if a host has already been connected to it and has forwarded packets. If not, a new MAC table entry is created for that host and the packet is forwarded to the node. The node performs match-action paradigm, to see if the packet header matches with any of the rules specified in the node. The node generally contains the rules for forwarding the packets between the hosts in the same node. Now in our case, suppose the packet header does not correspond to any of the rules, a default action of encapsulating the packets of the flow takes place and instantly, it is forwarded to the controller (table-miss). The controller on receiving the packet header, checks if it matches any of the nodes present in the node table that it maintains. When the appropriate node is found, it is forwarded to that node. The node on the other hand, unwraps it and sends it to the required switch. Thus this way of forwarding packets, avoids complex look-ups, flooding and delay in reactive flows.

So we have seen that the controller reactively installs flow rules into the nodes and removes them when necessary. Thus, the flow table from the TCAM memory of the switches (as in the conventional SDN) is removed and make the switches a mere forwarding device. Thus the TCAM memory of the switch cannot be exploited to forward fake packets. Node table is maintained by the controller and the flow table by the nodes.

When the packet header information (IP Address, MAC address) is to be matched with flow table, the following actions need to be performed according to its requirement:

- Forward the flow packet through an appropriate port to the switch. This is, if and only If the packet matches any host within that node.
- The packet is delivered to a dedicated channel from where it is encapsulated and forwarded to the controller. This is done for a packet header which does not match any host in the node.
- Move the packet to the cache memory. This is essentially done to curb the data to control plane saturation attack.

The controller maintains a centralized flow table to forward packets from the source node to the required destination node. The encapsulated packet header is matched with the node table by stating the IP address of the node to which it is to be delivered. This is more or less like matching packets in the traditional IP networks. The node here, maintains the starting address of the domain which helps inn identifying its connected switches and hosts. The node which receives the packet, unwraps the encapsulated packet to match the packet to the host based on the actions specified in the flow rules.

H. *Introducing QoS capabilities in nodes*

We know that the controller maintains the statistical data of the packet forwarding which takes place in the network. Based on the application required, the controller can introduce QoS capabilities in the network nodes. Since the network nodes contain the flow tables of all the switches connected to it, they can easily decode which switch data to forward first and give it the highest priority.

Data transfer between the node and the controller is divided into queues. Each queue can be given a different priority and can be allotted a different bandwidth for transferring data based on the requirement. This helps to transfer critical packets with high priority faster than the rest of the packets. These queue rules can be framed by the controller and sent to the nodes via policies as in MPLS (Multi Protocol Label Switched) networks.

III. Avoiding Vulnerabilities

A. *Enhanced Scalability*

The NCF framework can scale to larger networks easily since we offload the intelligence from the controller by proactively installing flow rules in the nodes. The controller frames rules by looking at the global network topology and by processing high level policy requirement of applications. We have seen that, the current SDN causes a considerable delay in processing new flow rules. Our framework constantly updates changes in the topology by using distance vector routing algorithm thus enabling the controller to change the flow rules in the node before the arrival of the packet in the node buffer. Thus any similar set of packets which arrive from the host can be processed instantly based on those flow rules. Switches act as mere forwarding devices containing the MAC address table and constantly updating changes. This will in turn help the nodes detect link failures. Hence an attacker can no longer exploit the vulnerabilities in the switches. The nodes in the network can be modified by the controller through a dedicated TTL/SSL connection. We may use redundant nodes in acritical network domain where the failure of nodes may incur huge costs.

B. *Updating at ease*

If a new application has to be introduced in the controller, whose operation is not supported by the NCF framework, then we needn't necessarily update all the switches. Updating individual switches may be a herculean task and may also lead to mismanaged updates. In this proposed framework, it will suffice if we update the nodes belonging to each network domain. This feature makes this system adaptable to changes dynamically and reduces the probability of mismanaged updates.

C. *Avoiding data to control plane saturation attack*

If an illegal or malicious user gains control of a host or if he is able to connect to a switch port through his Personal Computer, then he may try to saturate the controller with packets spoofed at random values. Node based cache memory gets flooded and in turn floods the controller thus using up all its resources. Since we are now aware of this attack we will try to mitigate it.

Real time detction of the attack

Firstly, we have to detect that an attack has been initiated. Anomaly based detection system, wherein, we keep track of maximum packet count in a particular time is generally easy to get around for an attacker who is willing to execute the attack gradually. We need to monitor the real time count of packets sent from the node to the controller. A spoofed packet's IP and MAC address isn't present in the MAC address table of switches. Hence constant fake addresses can be added which needs to be prevented. The spoofed packets are stored in the node's buffer before reactive flow rules are installed following a topology change. Therefore, we also need to keep a check on the packets queued at the buffer memory of the nodes.

Proactive flow rules and data plane cache

After the attack has been detected, the controller no longer installs reactive flow rules in the node based memory. The node based memory processes packets based on the last updated flow rules to forward the local traffic.

Consequently, the packets for which the flow rules don't exist in the node are forwarded to the centralized data plane cache memory. The cache memory does the job of connecting all the nodes to store packets during the saturation attack. The control plane fetches packets in the regular rate from the cache memory to either execute them or drop them in case if it turns out to be a spoofed packet. The controller does not process packets of particular type in order. Instead, the controller processes packets in a round robin fashion to prevent execution of spoofed packets all the time, as an attacker generally exploits a single agent type (TCP, UDP, and ICMP) to carry the attack.

State of defence

The controller, with the help of a statistical data tracks through which node the attacker has initiated the attack. Hence the particular node is made to go to a state of defense and it sends a message to all the switches to no longer accept new hosts. Thus even when fake MAC addresses are added, the switch does not breakdown and no topology changes effect the node. When the controller senses that the attack has stopped, it retains its normal state.

Thus we provide security against data to control plane saturation attack with a slight reduction in the performance of a particular node alone without affecting the whole network.

CONCLUSION

We thus design NCF (Node based Controlling and Forwarding) framework which divides the network into domains to enable fast packet processing and avoiding delays in the network. It provides us a secure framework to avoid vulnerabilities and glitches. Our framework can be emulated based on the design proposed to verify the efficiency of the results.

.

ACKNOWLEDGMENT

We thank Mr. Pownraj Jaeshu and Mr.Jaiganesh Balasubramanain for their valuable inputs and comments. We would like to thank our College and parents for supporting us in this endeavor.

REFERENCES

[1] Haopei Wang, Lei xu, Guofei Gu, FloodGaurd: A DOS attack prevention extension in software defined networks, In the Proceedings of 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 239-250

[2] Sungmin Hong,Lei Xu, Haopei Wang, guofei Gu, Poisoning network visibility in software defined networks: New attacks and countermeasures, www.internetsociety.org/default/files/10_4_2.pdf

[3] Minlan Yu, Jennifer Rexford, Michael J.Fredman,Jia Wang, Scalable Flowbased Networking wih DIFANE, SIGCOMM '10 Proceedings of the ACM SIGCOMM 2010 conference,pages 351-362

[4] Wolfang Braun,Michael Minth,Software Defined Networking using Openflow:protocols ,applications and Architectural design choices,www.mdpi.com/journal/futureinternet

[5] Amit Tootonchian,Yashir Ganjali ,Hyperflow: a distributed control plane for openflow,INM/WREN '10 Proceedings of the 2010 internet network management conferencce on research on enterprise networking , pages3-3

[6] Diego Kreutz, Fernando M.V.Ramos, Paulo Verissimo ,Towards secure and dependable sofware-defined Networks,HOTSDN '13 Proceedings of second ACM SIGCOMM workshop on hot topics in Software defined networks , pages 55-60

[7] Eugen Borcoci ,Control Plane Scalability in Software Defined Networks,Infosys 2014 conference April 25th ,2014 Charmonix France.

[8] M.Handley, O.Hudson and E.Kohler, "XORP: An open patform for network research ",proc.SIGCOMM workshop on hot topics in networking,Oct 2002

[9] Openflow: Innovate your network http://www.openflow.org

[10] Project Floodlight http://www.projectfloodlight.org/floodlight/

[11] S.shin,V.Yegneshwaran, P.Porras and G.Gu ,Avantgaurd: Scalable and Viglent switch flow management in software defined networks, In proceeddings of the 20th ACM conference on computer &communications security (CCS) ,2013