To Provide Privacy and Message Authentication Process Based Link Topology Discovery in Software Defined Network

Thota Renuka Kalyani[1], G. Sivalakshmi[2]

Final M.Sc Student[1], Lecturer[2]

[1, 2] M. Sc Computer Science, Chaitanya Women's PG College, Old Gajuwaka, Visakhapatnam Andhra Pradesh

Abstract:

Topology discovery is an essential service in software defined network and it underpins many higher layer services. When we refer to topology discovery, we actually mean link discovery, since the controller learns about the existence of network nodes during the Open Flow handshake. By implementing software defined network we can find routing and forwarding of transferred message. In the routing process we can find out link between source nodes to neighbour node. If the link in existing it can generate path and goes to next neighbour for finding link. If the link does not exist it will treat an attacker and source node will go to next neighbours. Like this we can find out routing from source node to destination node. After that we can transfer data from source node to destination before we can perform encryption process.. By performing encryption process we can convert plain format data in to cipher and send that cipher format to destination node. Before transferring cipher format data to destination node the source node will generate message authentication code for cipher data. The cipher format data and message authentication send to destination node. The destination node will retrieve cipher format data and message authentication code. The destination node will again generate message authentication code of cipher data and verify that code. If the code is verified it will perform the decryption process and get original plain format data. In this paper we are implementing secure link discovery protocol for finding routing and forward message. By implementing this protocol we can provide more security of transfer message or data and efficient routing of transfer message.

Keywords:

Put your keywords here, keywords are separated by comma.

I. INTRODUCTION

Software-defined networking is a new paradigm which revolutionizes network architecture through the introduction of a software-controlled, programmable forwarding plane. Traditional networking devices are typically autonomous in nature. Each device hosts its own operating system, runs distributed control-plane protocols and builds a local network state. The operating system, which is often proprietary, consults the local network state and configures specialized forwarding hardware through proprietary application programming interfaces (API) . SDN, on the other hand, eliminates these control-plane operations from network devices and moves the operating system to a logically centralized controller, also referred to as the network operating system. The controller exposes the network state learned from the forwarding devices to software-based network applications. Routing decisions are made by the applications and communicated to the controller, which in turn translates these decisions in to forwarding rules and programs the appropriate devices. Forwarding devices then perform packet header matching against these rules to determine the port on which to send a packet out. Communication between the network applications and the controller occurs over so called northbound APIs. Communication between the controller and forwarding devices occurs over southbound APIs. The forwarding devices constitute the data-plane, the controller constitutes the control-plane, and the networks applications form the management-plane . the difference between traditional networks and SDN networks.

SDN can also be defined in terms of three abstractions: forwarding abstraction, distribution abstraction, and specification abstraction.

• The forwarding abstraction allows network applications to make routing decisions without knowing any details of the underlying hardware. This is achieved through the use of open and standardized protocols for the communication with the forwarding devices.

• The controller implements the distribution abstraction. This abstraction is essentially responsible for two tasks. First, it is responsible for

installing forwarding rules on the network devices. Secondly, it gathers information about the forwarding layer and exposes this state information to network applications thereby allowing them to build a global network view.

• The specification abstraction allows network applications to express desired network behaviour without being responsible for the actual implementation of the behaviour itself.

SDN offers network operators many advantages. Decoupling the control and data planes allows the forwarding devices to be manufactured at lower costs since they no longer require the computing intelligence to perform control-plane processing . The centralized control allows the controller to maintain an up-to-date view of the full network topology. The exposing of this network state to software applications enables better informed forwarding decisions. Softwarization of the forwarding decisions accelerates innovation and service creation. Network operators no longer need to wait for standardization and implementation of new protocols. Instead, new functionality can be deployed as plug-and-play software modules.

II. RELATED WORK

One of the key roles of the SDN controller is to provide and maintain a global view of the network. The controller provides this view as an abstraction to the application layer, hiding a lot of the complexity of maintaining and configuring a distributed network of individual network devices. In this chapter, we focus on topology discovery, which is a critical service provided at the control layer of the SDN architecture, and which underpins the centralised configuration and management in SDN. The contribution of the research includes an analysis of the overhead of the current de facto standard for SDN topology discovery. We further propose an improved version and implement two variants of the basic idea. Our improved method achieves the same functionality, while reducing both controller CPU load and control traffic overhead by up to 40%. We present experimental results which demonstrate this.

One of the most important reasons to distribute the network control is based on the fact that one controller alone may not have enough capacity to manage the whole network, and therefore it could become a bottleneck in terms of processing power, memory, or input/output bandwidth. As explained in , in a centralized and reactive SDN network, scalability problems can be caused by flow initiation overhead or resiliency to failures. In large networks with a distributed control plane, these scalability problems may also arise, since controllers not only have to process requests coming from switches it is responsible for, but also requests sent from other controllers. As in a centralized SDN network, in a distributed SDN network, controllers have limited capacity of memory and CPU that can be saturated if the size of a network grows or if the switch load is not distributed homogeneously between the controllers, .In addition, increasing network traffic lead to a reduction of the available bandwidth in the links used by the control channels, limiting the switch-to-controller communication. This situation is critical in a reactive approach, given that the controller cannot do anything about the control link capacity as it cannot treat messages faster than it receives them.

Several approaches have been proposed to distribute the control plane across multiple controllers to improve the scalability of SDN, Kandoo , HyperFlow , and Onix , however, in these approaches the controller placement is not defined. Each one of those approaches distributes controller states differently. Kandoo distributes controller states by placing the controllers in two levels, a root controller and multiple local controllers. Local controllers respond to the events that do not depend on global network state, while the root controller takes actions that require a global network view. HyperFlow handles state distribution of the controllers through a publish/subscribe system based on the WheelFS distributed file system. Finally, controller state distribution in Onix is managed through a distributed hash table. In general, controller placement approaches are not concerned with the controller scalability, because they assume that commercial controllers are scalable in terms of capacity (quantity of flows processed per second). However, it has been demonstrated that, controller overload and long propagation delays among controllers and controllers-switches can lead to a long response time of the controllers, affecting their ability to respond to network events in a very short time and reducing the reliability of communication.

III.PROPOSED SYSTEM

This paper we are implementing secure link discovery protocol for finding route form source node to destination node and also provide security of transferring data. In the implementation of this protocol we can also provide authentication of transferred message. By performing message authentication we can identify the transferred message is corrupt or not. Because in the network we have attacker for corrupt data or loss the information. So that we can provide authentication of message and also provide security of transferred message. Before transferring message from source node to destination node the software defined

network controller information related to all nodes or switches in the network. The nodes information contains id, ip address of each node, port number and also maintain link status of each node in the network. The software defined network controller also send those links status all nodes in the network. So that by using link status we can identify routing from source node to destination node. After finding routing from source node to destination node the source node will perform encryption process for converting message into cipher format. The source node takes that cipher format data and generates message authentication code by suing message digest five algorithm. The completion message authentication process the source node will send cipher format data and authentication to destination node. The destination node will retrieve and perform the authentication process of message. If the message authentication process succeeds then the destination node will perform the decryption process and get original plain format data.

A) Nodes Initiation Process:

In this module every node will send request to software defined network controller for communication purpose. The SDN controller will accept request and send id, distance between nodes in the network. In the initiation process the software defined network controller also maintain information related to all nodes. The information of all nodes contains id of each node, ip address of nodes and port number. After sending the all nodes id and distances the software defined network controller will also send link status all nodes in the network. by using link status we can find out routing from source node to destination node. The implementation process of routing is as follows.

B) Shortest path route discovery:

In this module the source node will send request to software defined network controller and controller will check the all node link status. Based on link status of all nodes the software defined network controller will find the shortest route by calculating distance of all nodes in the network. in the routing process the software defined network controller will take all route information and check the if the link is exits its neighbour node of source node. If the link is exists it take those distance value and add into variable. Take another node of route and find out link is exists and take distance values add to variable. This process will repeat until end of the route. Take those routing distance value and compare for finding smallest values. Take that smallest value containing routing as shortest route and transferred message through that path. After completion of routing process the source node will perform the encryption process.

C) Encryption process:

In this module the source node will transferred message to destination node. Before transferring message the source node perform the encryption process. The implementation process of encryption is as follows.

1. Input the text .

2. Convert the previous text to ASCII code.

3. Convert the previous ASCII code to binary data.

4. Find out One's complement of the previous binary data.

5. After converting one's complement that data can be generate in grid with size of 32*32.

6. After that we can shift outer circle in clock wise and inner circle shift anti clock wise.

7. The completion of shifting operation we can get that data and send to destination node.

After completion of encryption process the source node will generate authentication code and send to destination node. The implementation process message authentication code is as follows.
Message Authentication Code Process:

In this module the source node will generate authentication code for cipher format data. The generation of authentication code is as follows.

D) Algorithm: authentication code of message or data

Input: The plain text

Output: authentication code message and append to message

E) Procedure:

Msg_Len= Calculate length of (message)

Block_len=length of block is 128 or 256 or 512

Res_bits---->take the 16 bits are reserved bit

P=Msg_len%Block_len;

Q=Block_len-(Msg_len+Res_bits)

If(Q>0)

F_1→append the Q zeros to end of file

Else if(Q<0)

R←Block_len+Q

F_l← append R zeros to end of the file

F_l ← the reserved bits are append to end of file

The following steps are generating signature of file as

Len← calculate length of file after append zero to end of file

Count← len/Block_len

For I ←1 to count

S←0

S←reverse[$\sum_{a}^{block\_len}$ (A®B)V(A^B)]

Where B←Ascii value of character (A)

Sig←sig+to_binary(S)

F_N←cipher+sig

The source node takes those cipher format data and authentication code send to destination node through shortest path. The destination node id will retrieve cipher format data and authentication code again generates authentication code for cipher format data. The generation of authentication code is same as message authentication process. After completion of verification process the destination node will perform the decryption process. The implementation of decryption process is as follows.

1. The receiver will get cipher format data and generate grid with size of 32*32.

2. After completion grid generation we can perform clock wise shift of outer circle and anti clock wise of inner circle of grid. That way we can perform all circle shifting and get shifted cipher format data.

3. Take that shifted data and perform the once complement for completion length of message.

4. After completion of once complement we can convert that data into ascii format.

5. After that data can be convert into character.

6. After converting character we can get original plain format data.

By implementing those concepts we can improve performance for finding routing and also provide more security of transferred message.

IV. CONCLUSIONS

In this paper we are proposed secure link topology discovery for transferring data from source node to destination node. Before transferring data the server will build network and the network will contain link state of each node in the network. After that the server will send all remaining nodes information to all nodes. The completion of network building source node will enter transferred message and performing the encryption process. By performing encryption process the message will be convert into cipher format. Take the cipher format and generate authentication code for that cipher format data. After completion of message authentication code and source node will send cipher format data and message authentication code to destination node. Before sending information the server will find out shortest route and also find out each node link status. if the link status is available in the every node and sever will send information through shortest route. The destination node will retrieve cipher format data and message authentication code. After that the destination node will perform the decryption process and get original plain format data without corrupting of cipher format data. If the cipher format data is corrupted stop the decryption process and will not get any message. By implementing those concepts we can provide more security of transferred message and also generate an efficient routing discovery by using link topology.

REFERENCES

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn," Queue, vol. 11, no. 12, p. 20, 2013.

[2] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in IEEE ICSPCS, 2015.

[3] GENI Wiki. [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 3, Jul. 2008.

[5] Open Flow Standard. [Online]. Available: https://www.opennetworking.org/sdn-resources/onfspecifications/openflow

[6] IEEE standard for local and metropolitan area networks–station and media access control connectivity discovery," IEEE Std 802.1AB-2009 (Revision of IEEE Std 802.1AB-2005), pp. 1–204, Sept 2009.

[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks.

[8] Open vSwitch. [Online]. Available: http://openvswitch.org

[9] Scapy Library. [Online]. Available: http://www.secdev.org/projects/scapy/doc/usage.htm

[10] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication," IETF RFC 2104, pp. 1–11. , February 1997.

[11] S. Turner and L. Chen, "Updated security considerations for the md5 message-digest and the hmac-md5 algorithms," IETF RFC 6151, 2011.

[12] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," in NDSS'15, February 2015.