A Novel Efficient Host Detecting Scheme in SDN

Yanwei Xu
Shanghai Engineering Research Center for Broadband Networks and Applications
Shanghai, China
ywxu@bnc.org.cn

Xiaoyuan Lu
Shanghai Engineering Research Center for Broadband Networks and Applications
Shanghai, China
xylu@bnc.org.cn

***Abstract*—Data centers with millions of physical and virtual end hosts are now the basis for many Internet and cloud computing services. Hence, it is no longer practical to employ the traditional flooding method for host detecting in such large networks. In order to improve the efficiency of host detecting among millions of hosts, we present a novel Software Defined Networking (SDN) based broadcast scheme in this paper. In most of the existing SDN technologies, the centralized Controller usually outputs messages to every port of all switches for broadcasting. However, in our scheme, a broadcast tree is previously created and corresponding instructions are installed into the switches as flow entries by the Controller. Hence, the Controller only needs to send a message to one switch, and then the informed switch will automatically spread the message to entire network along the broadcast tree. Therefore, the heavy broadcast task is transferred from the Controller to the distributed switches, which greatly alleviates the burden on the Controller and reduces the interactions between the Controller and switches. Moreover, a prototype is developed and the experimental results prove the good efficiency of the presented broadcast scheme.**

Keywords—Broadcast; SDN; Host detecting scheme

I. INTRODUCTION

The big data and cloud computing that require efficient cooperation among millions of hosts are widely developed and used in IT, industry 4.0 [1], entertainment and education, etc. Nevertheless, many existing technologies [3, 4, 5, 6] are not suitable for such large scale networks since 1) they have slow convergence speed and low forwarding efficiency; 2) or they usually use flooding method to broadcast messages, which is prone to endless loop and excessive resource consuming. However, Software Defined Networking (*SDN*) [7, 8] is emerging as a promising solution to those problems in recent years, which is based on some key attributes such as decoupling control plane (a logically centralized control software module named as Controller) from data plane, supporting uniform vendor-agnostic interfaces for forwarding devices (i.e., *OpenFlow* [9, 10, 11]) and having the ability to virtualize the underlying physical network. Because the Controller is able to learn the whole topology and control all machines (switches and hosts) in the network, the broadcast directions and paths can be ordered, which results in no broadcast storm. However, in most existing SDN technologies (such as OpenDayLight (*ODL*), a typical existing OpenFlow solution [12]), if the Controller needs to resolving ARP requests, it always adopts the way of spreading the message to each port of all switches, which is time-consuming and inefficient.

In order to overcome the disadvantage, we present a novel efficient SDN based broadcast scheme in this paper. In the proposed scheme, a broadcast tree is created in advance, which means the path between any pair of switches has already been constructed previously and thus no broadcast loop will appear. Next, if the Controller wants to broadcast a message, it only needs to send the message to one switch, and then the informed switch will automatically spread it to other switches along the broadcast tree like radiation. Hence, our solution transfers the broadcast tasks from the centralized Controller to distributed switches, which alleviates the heavy burden on the Controller, reduces the load on the control planes between the Controller and switches, and results in improving the broadcast efficiency.

The rest of the article is structured as follows. Section 2 lists the related work. Detailed description of the presented broadcast scheme is shown in Section 3. And then, Section 4 gives the experiments that can fully demonstrate the efficiency of the proposed scheme. Finally, we summarize the article with conclusions and future research directions.

II. RELATED WORK

Because the ARP is known as the major source of more than 88% of all broadcast traffic according to [13, 14], in this paper, we take the ARP resolving as the example to explain how the proposed broadcast scheme works and show its good broadcast efficiency.

There are always two operation paradigms in many SDN solutions, such as NOX/C++ Controller [15], the POX/Python Controller [15], the Trema/Ruby Controller [16] and the Floodlight/Java Controller [17], while dealing with ARP process: reactive and proactive. In the reactive approach, the first packet of a flow will trigger the Controller to broadcast the ARP request to look for the location of the destination host and insert flow entries into each OpenFlow switch. This approach may utilize the existing flow table memory well, but the Controller should afford heavy work burden and the broadcast efficiency is much low. In the proactive approach, the end hosts will provide the IP-to-MAC mapping information to the Controller proactively, which makes the Controller pre-populate its database. As a result, the Controller can process the ARP request without the need of implementing broadcast. However, if there are massive amount of end hosts changing their states (migration, active or down) in a short time, such as ARP attack, this proactive manner will bring disaster for the Controller and SDN network.

III. The Proposed Network Broadcast Scheme

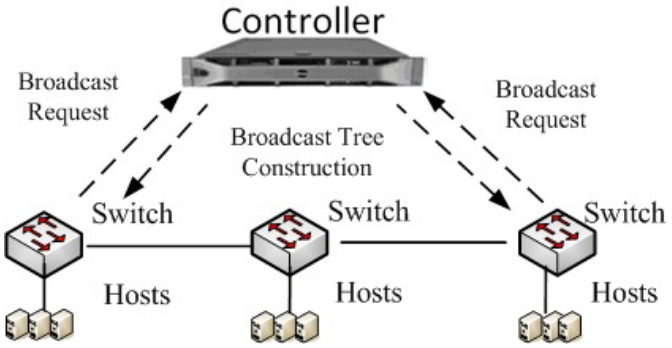


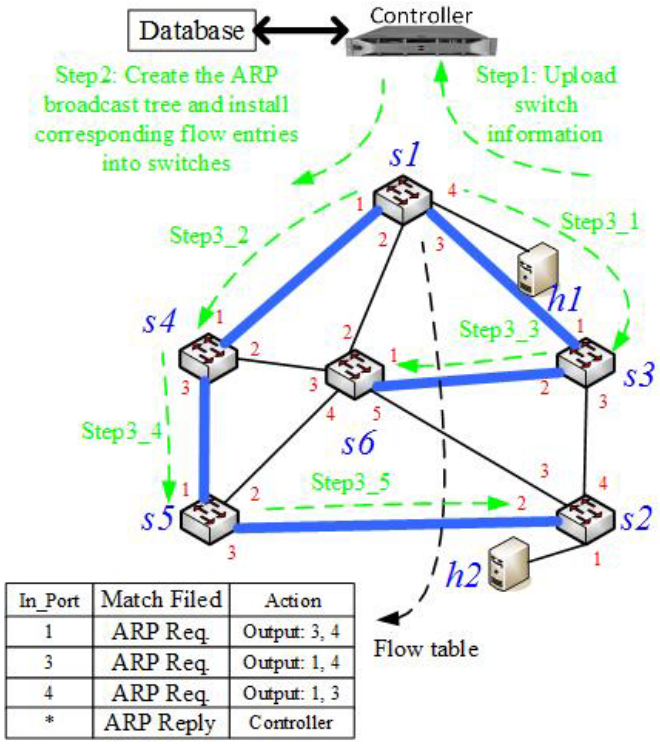
Fig. 1. The architecture of a SDN based network.

In SDN, the most important component is the centralized Controller that controls all switches based on OpenFlow protocol. Fig. 1 shows the basic architecture of a SDN based network. Considering the advantages and disadvantages of the reactive and proactive paradigms, a hybrid technique is used by us. We employ the proactive way to construct a broadcast tree on the switch level in advance, and then the reactive method is adopted to solve the ARP requests from end hosts. However, compared with many existing SDN solutions, with the help of the well-designed broadcast scheme based on the previously built broadcast tree, the broadcast efficiency of the ARP resolving in our scheme is improved dramatically.

In our proposed scheme, the Controller is mainly responsible for the creation of the broadcast tree, the configuration of broadcast flow tables according to the constructed broadcast tree, and the installation of broadcast flow tables into switches. On the other hand, each switch is mainly responsible for spreading messages to two kinds of machines: its neighbouring switches along the broadcast tree and the hosts that directly connect to it under the instruction of its flow table.

There are three working steps in the proposed broadcast scheme, and the details are described as follows.

Step 1: Once the network boots up, each switch employs the Link Layer Discovery Protocol (*LLDP*) [18] to investigate its neighbourhoods. And then the switch information (e.g. the port attributes and the connection states among switches) will be uploaded to the Controller via OpenFlow protocol. Hence, the Controller knows the entire network topology.



In_Port	Match Filed	Action
1	ARP Req.	Output: 3, 4
3	ARP Req.	Output: 1, 4
4	ARP Req.	Output: 1, 3
*	ARP Reply	Controller

Fig. 2. The ARP broadcast tree and corresponding flow entries in the proposed scheme.

Step 2: After Step 1, the Controller starts to adopt the Prim algorithm [19] to create Minimum Spanning Tree (*MST*) on switch level, and then takes it as the broadcast tree of the network. For instance, the broadcast tree in Fig. 2 is *s6* ↔ *s3* ↔ *s1* ↔ *s4* ↔ *s5* ↔ *s2* (the blue thick lines).

Afterwards, the Controller is able to configure the broadcast flow tables according to the constructed broadcast tree as soon as possible. In Fig. 2, the bottom table is the broadcast flow table used by *s1*, and the red value indicates the port number of each switch. In the broadcast flow table of *s1*, the first column indicates the input port of a broadcast message; the last column lists the corresponding action, which is to spread the broadcast message via the ports that directly connect to the hosts under *s1* or the neighbouring switches of *s1* along the broadcast tree. For example, if a message arrives at *s1* from port1, then *s1* will send the message to its port3 (connects to *s3*) and port4 (connects to *h1*).

Lastly, the Controller installs the broadcast flow tables into all switches. For example, the bottom table in Fig. 2 is installed into *s1*. It should be noted that the database of the Controller has mastered the information of switches since the broadcast tree has been proactively constructed on the switch level in advance, but been aware of nothing about the hosts until now.

Step 3: This step focuses on the process of ARP resolving. As we said, the database of the Controller does not store the IP-to-MAC address mapping information of hosts before this step. Hence, if *h1* wants to send packets to *h2* (Fig. 2), the Controller has no idea where *h2* is, but has to broadcast the ARP request of *h1*.

At that time, in traditional SDN based technologies (e.g. ODL), the Controller always sends the ARP request to each port of all switches in a flooding manner, which puts a heavy burden on itself and severely affects its working speed. However, in our scheme, the above disadvantage can be overcome since the broadcast flow tables are configured based on prepared broadcast tree and thus the broadcast load originally put on the Controller has been moved to the distributed switches. We can take Fig. 2 as the example to explain the procedure in detail.

In our scheme, due to the fact that the database of the Controller is helpless at the beginning, the Controller has to send a *packet_out* message to *s1*. Then *s1* will spread the message to its port3 and port1 according to the second flow entry installed by the Controller in Step 2. Thus *s3* can receive the message from port1 and send it to next switch *s6* through port2 (Step3_1). Similarly and simultaneously, *s4* can receive the message from port1 and send it to next switch *s5* through port3 (Step3_2). Afterwards, *s6* can receive the message from port1 (Step3_3), *s5* can receive the message from port1 and send it to next switch *s2* through port2 (Step3_4). At last, *s2* can receive the message from port2 and send it to the destination host *h2* through port1 (Step3_5). As a result, the message is broadcasted to entire network and finally *h2* will respond as soon as it receives the message. After acquiring the ARP response, the Controller learns the IP-to-MAC address mapping information of *h1* and *h2*, and then it stores the information in its database. Therefore, the Controller does not need to implement broadcast in the future if other hosts want to send packets to *h1* or *h2*. Instead, the Controller can directly reply the ARP request according to its database.

Since several switches (*s3* and *s4*) may spread the message in parallel like the radiation due to the two broadcast directions in the broadcast tree, the broadcast speed of our scheme is much faster than those existing SDN solutions which use the Controller to send ARP request to each port of all switches one by one.

IV. Experiments

Because our scheme has employed the switches to take the responsibility of broadcasting messages, the heavy broadcast task is moved from the centralized Controller to distributed switches. Therefore, compared with other existing SDN based technologies, such as ODL, our scheme is able to improve the work efficiency of the Controller significantly.

For fully proving this point, we implement a prototype with four network scenarios, which contain 50, 100, 150 and 200 OpenVSwitch, respectively, and each switch is in charge of 100 KVM virtual machines under linux. To prove our proposed scheme can be used in general network, the network topology of the prototype is random. The centralized Controller is developed based on the ODL platform. Because the ARP resolution is a typical broadcast process, we choose to observe the ARP resolution time performances between ODL and our scheme in the four network scenarios.

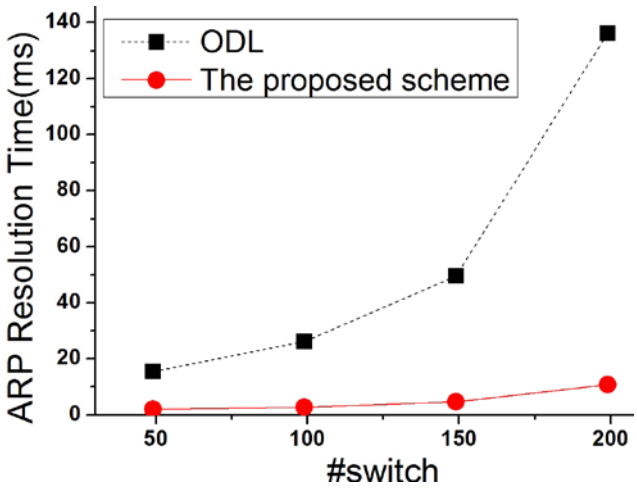


Fig. 3. The comparisons of the ARTs between ODL and our proposed scheme with different network sizes.

The ART of ODL , as shown in Fig. 3, is much longer since the Controller must flood the searching message to each port of all switches one by one, which is very time-consuming. Unsurprisingly, our proposed scheme achieves much lower ART since the distributed switches take the responsibility of spreading messages to the entire network along the prepared broadcast tree in such a radiation manner. Besides, although the ART of our scheme grows with the increment of switches, the growing speed is very slow, making the ART differences between the ODL and our proposed scheme become larger with the growth of the number of switches. Therefore, from Fig. 3, we can see the larger the network size is (implies the bigger number of switches), the higher broadcast efficiency our scheme has compared with other SDN based technologies (such as ODL).

V. Conclusion

In this paper, we have proposed a novel efficient SDN based broadcast scheme for host detecting. Different from other SDN solutions, the central Controller in our proposed scheme creates the broadcast tree in advance, and then the distributed switches is able to take charge of spreading message to entire network along the broadcast tree. A prototype is developed and the corresponding experimental results have proven the good efficiency of the proposed scheme. Although we use ARP resolving as the example to illustrate the broadcast scheme in this paper, in fact, this scheme is suitable for all kinds of broadcast tasks since the switches take the broadcast responsibility and spread the messages in parallel like the behavior of radiation. However, there is still space for us to improve the broadcast scheme further in the future. For example, we can optimize the way of creating broadcast tree and corresponding flow tables so as to make the scheme more intelligent and efficient.

Acknowledgment

We would like to thank the 863 plan project under the grant No 2013AA013505.

REFERENCES

[1] C.T. Yen, Y.C. Liu, C.C. Lin, C.C. Kao, W.B. Wang, and Y.R. Hsu, "Advanced manufacturing solution to industry 4.0 trend through sensing network and cloud computing technologies," In Automation Science and Engineering (CASE), pp.1150–1152, 2014.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "Ascalable, commodity data center network architecture," In Proceedings of the ACM SIGCOMM 2008 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Seattle, WA, USA, August 17-22, 2008, pp.63–74, 2008.

[3] G. Singh and A.J. Bernstein, "A Highly Asynchronous Minimum Spanning Tree Protocol," Distributed Computing, 8(3), pp.151-161, 1995.

[4] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," In Proceedings of the ACM SIGCOMM 2009 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Barcelona, Spain, August 16-21, 2009, pages 39–50, 2009.

[5] M. Scott, A. Moore, and J. Crowcroft, "Addressing the scalability of ethernet with moose," In First Workshop on Data CenterCConverged and Virtual Ethernet Switching (DC CAVES), ITC 21, Paris, 2009.

[6] H. Wang, "TRILL-based Large Layer 2 Network Solution," White Paper, 2012.

[7] O. N. Foundation, "Software-Defined Networking: The New Norm for Networks," White paper, 2013.

[8] A. Voellmy and J. Wang, "Scalable software defined network controllers," In ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland - August 13 - 17, 2012, pages 289–290, 2012.

[9] P. Dely, A. Kassler, and N. Bayer, "Openflow for wireless mesh networks. In Proceedings of 20th International Conference on Computer Communications and Networks, ICCCN 2011, Maui, Hawaii, July 31 - August 4, 2011, pp.1–6, 2011.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G.M. Parulkar, L.L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks. Computer Communication Review, 38(2), pp.69–74, 2008.

[11] R. Sherwood, M. Chan, G. A. Covington, G. Gibb, M. Flajslik, N. Handigol, T. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. M. Parulkar, "Carving research slices out of your production networks with openflow" Computer Communication Review, 40(1), pp.129–130, 2010.

[12] OpenDayLight. http://OpenDayLight.org

[13] K. Elmeleegy and A. L. Cox, "EtherProxy: Scaling Ethernet By Suppressing Broadcast Traffic," IEEE INFOCOM, pp.1584-1592, 2009.

[14] A. Shpiner, I. Keslassy, C. Arad, T. Mizrahi, and Y. Revah, "SAL: Scaling Data Centers Using Smart Address Learning," Technical Report TR 14-02, Technion, 2014.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," SIGCOMM Computer Communication Review, 38(3), pp.105–110, 2008.

[16] NEC, "Trema Openflow Controller," Last accessed, Aug 2012. [Online]. Available: http://trema.github.com/trema/

[17] D. Erickson, "Floodlight Java based OpenFlow Controller," Last accessed, Aug 2012. [Online]. Available: http://floodlight.openflowhub.org/

[18] M. Yokohata, T. Maeda, and Y. Okabe, "An extension of the link layer discovery protocol for on-demand power supply network by poe," In Advanced Information Networking and Applications Workshops, pp.1612–1616, 2013.

[19] B. Chen, F. Wei, J. Pan, and Y. Xia, "The minimum spanning trees of trna sequences based on prim's algorithm," In the Fifth International Conference on Natural Computation (ICNC), pp.176–179, 2009.