A POX OpenFlow Loop Solution for Mininet Network Emulations

Silvan Streit
Radiocommunications Laboratory, Physics Department,
Aristotle University of Thessaloniki,
Thessaloniki, 54124,
Greece
sstreit@physics.auth.gr

Christos Kalialakis
Radiocommunications Laboratory, Physics Department,
Aristotle University of Thessaloniki,
Thessaloniki 54124,
Greece
kalialakis@ieee.org

***Abstract*—Mininet is an open source platform for network emulation. Several network architectures can be implemented and tested in conjunction with an OpenFlow controller. In this work, a solution to the network loop problem is presented by writing in Python a custom component for the POX OpenFlow controller platform. The Google WAN network is used as a test network structure to highlight the proposed implementation. The algorithm proposed is generic and can be used in other OpenFlow controllers which use different Application Programming Interfaces.**

Keywords—Mininet; SDN; Python; OpenFlow; Google; Network Loops

I. INTRODUCTION

The major trends in telecommunications networking[1] are, on the one side the increasing degree of dependence on software, a process recently termed softwarization [2] and the vast complexity. Software Defined Networking (SDN) in particular appears as an approach that goes along this roadmap [3]-[4].

On the complexity side, tools that allow network emulation and test before implementation are highly desirable. On the software side, ease of experimentation with controller structures that can help reduce hardware dependencies is highly sought. Mininet [5] has surfaced as a popular approach to achieve both goals especially when combined with OpenFlow [6]. Several controllers are available such as POX, NOX, Floodlight and others. For this study, POX was used because it has a Python language interface and is used widely for research [6]. OpenFlow in general has attracted substantial interest from industry [7]-[8].

In this work, the focus is on the network loop problem. Algorithms such as the Spanning Tree Protocol and Shortest Path Bridging are used in commercial switches to provide a logical loop free operation in networks with physical loops. In this work, a simple software solution is demonstrated. After providing some background on Mininet in Section II, the Google WAN [9] is used as the network structure (Section III) to discuss the solution to this problem (Section IV). The solution is implemented by writing a custom component for the POX controller platform using the scientific computing language Python.

II. MININET BACKGROUND

A. Mininet

Mininet is an open source network emulator that runs on a Linux virtual machine. It can be combined with other software components such as a remote OpenFlow controller. Traffic analysis can be performed with packet sniffers such as the open source Wireshark [10]. There are three planes in the SDN architecture [11]; data, control and application (Fig.1). In this paper, the interest is in the bottom two planes.

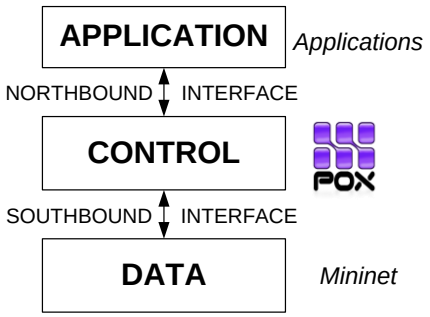

Fig. 1. Mininet and POX OpenFlow in SDN architecture.

B. Network Components

In terms of fundamental components a network can be described as a collection of hosts, switches and controllers connected by links. Simple topologies [12] can be used as test structures for customized functions of OpenFlow controllers. For example, a linear network and a single switch network are shown in Fig.2 and Fig.3 respectively. Larger networks with every possible topology can be created by defining all components and their connections utilizing the high level Python Application Program Interface (API).

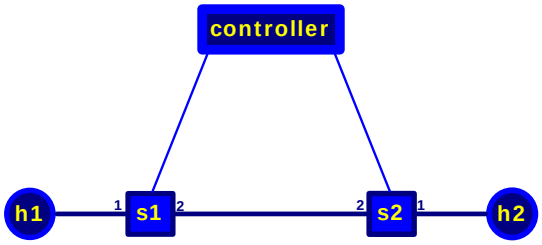

Fig. 2. A Linear Network with two hosts and a controller

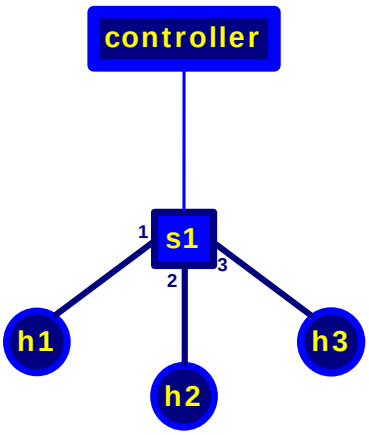

Fig. 3. A Single Switch Network with three hosts

III. The Loop Problem: Google WAN as Case Study

As a test structure in this work, the Google OpenFlow WAN[9] is used. This network comprises of 12 nodes as shown in Fig.4. These nodes where simulated as OpenFlow switches with two hosts connected to each of them (File google-topo.py). Path loops are also depicted in Fig.4.

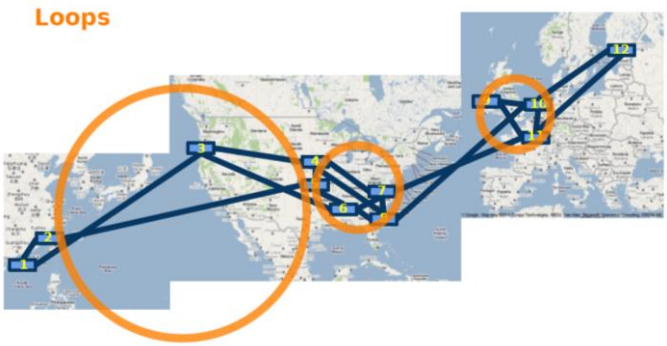

Fig. 4. Google WAN Network (adapted from [9]).

If a controller is used in a straightforward fashion (Fig.5) using a typical hub like or learning switch behavior, packets will be caught in the loops. Those packets will be forwarded indefinitely, causing the network eventually to a denial of service. Therefore a new adjusted controller has to be employed which is aware of the loops and prevents packets from being trapped.

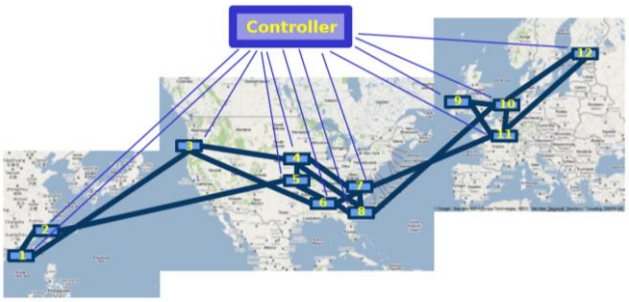

Fig. 5. Google WAN network using an OpenFlow controller with paths from the controller to each host (adapted from [9]) .

IV. A Loop Solution

A solution to the problem described in the previous section requires a new custom component for the POX controller platform. The algorithm for the new component operation is shown in Fig.6.

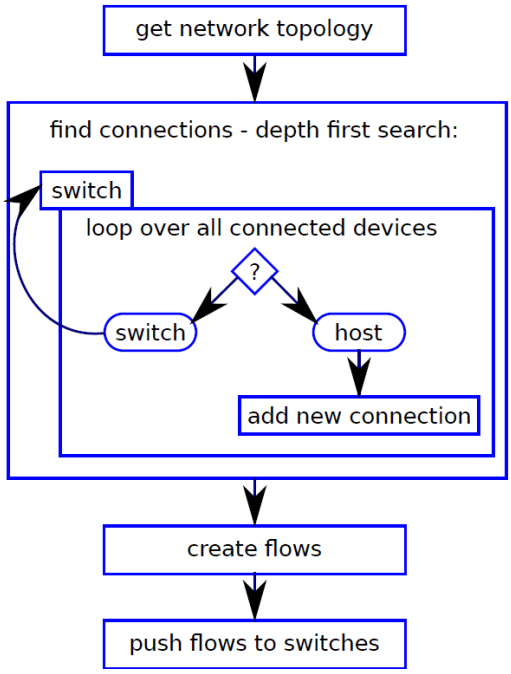

Fig. 6. Flow chart of the controller operation algorithm.

The flow chart (Fig. 6) is general enough and could be readily transferred to other OpenFlow controllers. In the case of POX in this work, it was realised in Python (file custom-pox.py).

The controller is fed with the topology of the network and all associated components; practically a list of all switches and of all hosts but with the addition of direct links and the involved ports of the devices on each end of the link.

In the initialization phase, the controller is instructed to map the network paths. In order to achieve this, a recursive

graph search is conducted by searching for paths between all hosts. A list of visited nodes is recorded to prevent loops and multiple visits. Every time a new host is visited, a new connection is created with the new host at the end and all open connections from the hosts visited so far are copied to the final list of connections, connected with the new host. Every time a new switch is visited, a recursive search of its connections is triggered which returns a list of all connections to be reached via this switch. Again all the new connections which are returned are appended to the final list of connections connected to the open connections found so far. Recall that the involved ports of all the devices in the connections are also recorded in the list of connections. This allows the transformation of the connections to flow control entries which can be pushed to the switches.

After all connections are found the list of flows is created. Effectively, each switch searches for itself in each connection and locates its port mappings corresponding to the reachable hosts. These port mappings are then saved as flow entries in the switch object inside the controller. Once the switches connect to the controller, the flow entries are pushed to the switches which allow proper package handling without further need of controller involvement.

It should be noted that only packets to known hosts can be forwarded over the network. In this way, only paths which the controller knows are taken. On one hand, this approach allows the proper handling of loops but on the other hand no new hosts can be addressed which the controller is not aware of. Recall that hosts inside a local network are directly addressed by their Medium Access Control (MAC) address and the Address Resolution Protocol (ARP) is always used to resolve the host's MAC address given its IP. Normally the host with the target IP address responds with a broadcast containing its MAC address. The problem in the loop network is that broadcasts cannot be forwarded. Therefore, a different way is needed to answer to an ARP broadcast in this context. Instead of forwarding the broadcast message and wait for the correct host to answer, the controller generates the response and passes it back to the host. The OpenFlow controller POX allows handling of such low level problems with a high level API. To this mean, POX offers a module called arp_responder. Together with a static ARP table of all hosts, ARPs can now be handled without the need of forwarding the packages over the network. If a host wants to know the MAC of another host, an ARP request is sent to the connected switch. As the destination of the broadcast is unknown, the switch forwards the ARP request to the central controller. The arp_responder looks up the corresponding MAC of the other host and creates an ARP response containing the MAC of the other host which is passed back to the host through the connecting switch.

In order to test the controller and simplify the traffic analysis, a circle topology (Fig.7) was first implemented in Mininet to emulate a single loop in the Google WAN (File circle-topo.py).

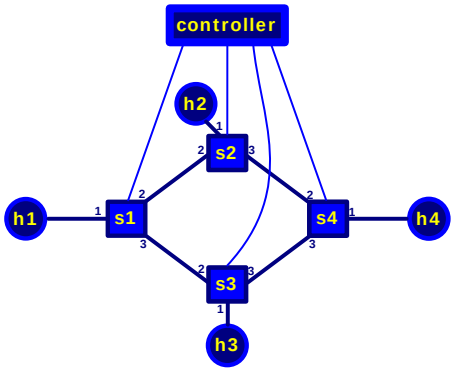

Fig. 7. A circle topology implemented in Mininet to address the loop problem. Switches have been added before each host.

Having tested the solution on the single loop of Fig.7, the full network topology of Fig. 5 can now be tackled. 276 host-to-host paths are discovered using the recursive graph. With the custom loop controller in place, the infinite loops are dissolved completely (File custom-pox.py).

Please note that all the Python files necessary for the loop avoidance implementation and testing (circle-topo.py, google-topo.py, custom-pox.py) are available for download as open source at the github website in the following URL; https://github.com/mocast/OpenFlow_Loop_Solution .

V. Conclusions

A solution implemented in Python has been proposed to overcome the infinite loop problem by creating a custom component for the POX controller platform. The solution was demonstrated on an emulated Google WAN Network. Further work is needed in order to increase the algorithm efficiency and to automate arbitrary network topology detection. The proposed solution is generic enough that can be applied to other available OpenFlow controllers with different APIs.

Acknowledgment

The first author was supported by the Deutschlandstipendium and the Erasmus+ mobility grant for study at Aristotle University of Thessaloniki coming from the Ludwig-Maximilians-Universität Munich, Germany.

References

[1] Shenker, S. "The Future of Networking, and the Past of Protocols", *Open Networking Summit*, 2011

[2] Galis, A. et al. "Softwarization of Future Networks and Services - Programmable Enabled Networks as Next Generation Software Defined Networks," 2013 IEEE SDN for Future Networks and Services (SDN4FNS), pp.1-7, 11-13 Nov. 2013

[3] IEEE Software Defined Networks, online at http://sdn.ieee.org/

[4] An IEEE Think Tank on SDN / NFV (Softwarization), available online at ieee-sdn.blogspot.com

[5] Mininet, *An Instant Virtual Network in your Laptop (or other PC)*, online at mininet.org

[6] Rodrigues Prete, L. et al, "Simulation in an SDN network scenario using the POX Controller," 2014 IEEE COLCOM Conference , pp.1-6, 4-6 June 2014

[7] Levy, S., "Going With the Flow: Google's Secret Switch to the Next Wave of Networking", Wired, April 17, 2012.

[8] Neagle, C. "HP takes giant first step into OpenFlow: HP is announcing its first effort to support OpenFlow standard on its Ethernet switches". Network World, February 2012.

[9] Hoelzle, U., "Google @ Open Flow WAN", *Open Networking Summit* 2012, available online at http://www.opennetsummit.org/archives/apr12/hoelzletue-openflow.pdf

[10] Wireshark, Stable Release 1.12.3, available online at https://www.wireshark.org.

[11] Open Networking Foundation, SDN architecture , SDN ARCH 1.0, June 2014.

[12] Kaur, K., Singh, J. and N. Singh Ghumman. "Mininet as Software Defined Networking Testing Platform", International Conference on Computing, Communication & Systems, 2014.