Software Defined Networking: Problem Statement

Loic Niragire

TIM-7010 Computer Networks & Mobile Computing

Dr. Phillip Davis

Table of Contents

Introduction

SDN emerged as a solution to administrate network switches by layering the control plane. Whereas the data plane is built on solid abstractions using clearly defined layers. The control plane handles various mechanisms and goals such as routing and isolation but lacks modularity and proper abstractions. To this end, SDN introduced two control plane abstractions, the Global Network View and the Forwarding Model. The Global Network View generates the network's graph model, or topology, from gathered physical network details through the Network Operating System. On the other hand, a control program takes the produced graph model as input and computes the forwarding state that every network switch should have, thereby making SDN directly programmable. Figure one below depicts the layers discussed above.

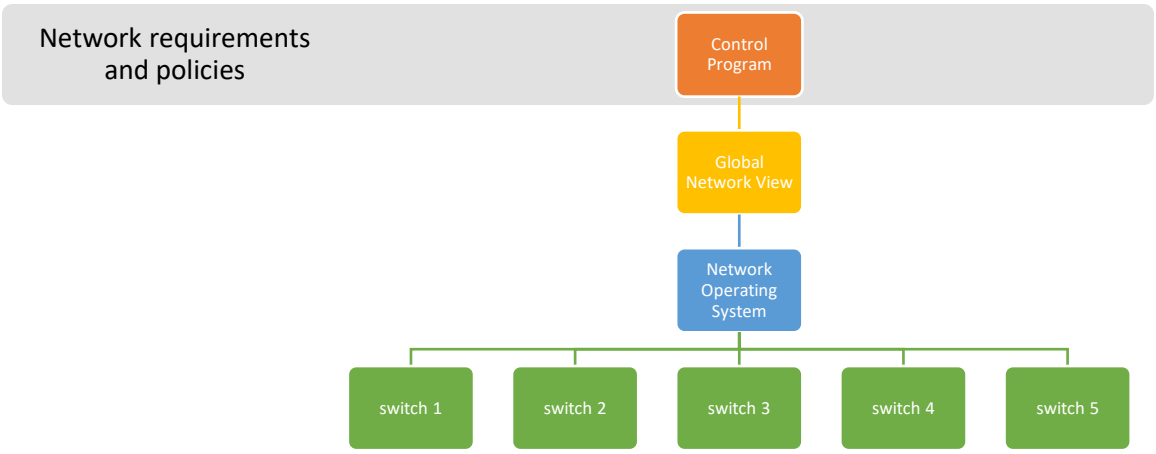

Figure 1 Control Plane layers

Operators express network requirements and policies through control programs that communicate desired behavior to each node in the Global Network View. This coupling makes the control program responsible for adjusting to changes in the network topology. SDN solves this problem by introducing a network virtualization layer between the control program and the network operating system. This network virtualization layer comprises a virtual topology and a network hypervisor that compiles the operator's requests into the network operating system. Therefore, the network hypervisor takes on the responsibility of handling network topology changes.

Problem Statement

The integration of SDN and IoT presents a security challenge because switches rely on limited-sized expensive flow tables to instruct them how to direct network traffic. With the enormous amount of data and traffic generated by IoT devices, the risk of overloading these flow tables becomes inevitable. Consequently, this introduces the potential for pocket forwarding failures at the network switch (Sood, Yu, & Xiang, 2016). A protocol known as OpenFlow provides a software-based access to flow tables. Where an OpenFlow switch is composed of one or more flow tables and a group table. Each switch communicates to one or multiple controllers via message exchange over a secure OpenFlow channel. Moreover, this communication can either be initiated by the switch or the controller. With the initial handshake established, the switch securely sends each packet without a flow entry match to the controller for a forwarding decision. To be noted, by default switches drop all unmatched packets unless configured to defer to the controller. Assuming the later, subsequently, the controller inserts a routing message into the switch's flow table, indicating packet forwarding action (Cisco, 2021). Given that flow tables are limited size, there is a security risk of packet loss if the controller fails to insert flow entries for unmatched packets.

This problem exists because, under SDN, the switch does not contain packet forwarding logic. Instead, it maintains a look-up table containing packet identifiers and corresponding forwarding action. It is the controller's responsibility to manage content for this look-up table (Naous, Erickson, Covington, Appenzeller, & McKeown, 2008). Alternatively, the controller can opt to update a flow entry instead of inserting a new flow upon unmatched packet. Each flow entry is a 10-tuple that can be matched exactly or using wildcards for fields. Figure 2 below illustrates Type-0 switch available flow fields.

Figure 2 Type-0 switch flow fields

A typical flow table can hold more than 2,000 flow entries, such as those built on the Broadcom chipset. Meanwhile data centers can reach upwards of 10,000 flows per server rack. To efficiently handle flow table capacity limitation, current solution involves a strategy to evict inactive flow entries. Which inevitably requires identifying such entries and that is a challenging task given the time constraint for the network to remain responsive (Yang, Riley, & Blough, 2020). Ongoing research on this problem explore use of machine learning to identify active and inactive flows in a network.

Reference List

Cisco. (2021, April). OpenFlow. Retrieved from cisco.com: https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst2960xr/software/15_2_7_e/configuration_guide/b_1527e_consolidated_2960xr_cg/openflow.pdf

Naous, J., Erickson, D., Covington, A. G., Appenzeller, G., & McKeown, N. (2008, November). Implementing an OpenFlow Switch on the NetFPGAplatform. ANCS.

Sood, K., Yu, S., & Xiang, Y. (2016). Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review. IEEE Internet of Things Journal, 453-463.

Yang, H., Riley, G. F., & Blough, D. M. (2020, February). STEREOS: Smart Table EntRy Eviction for OpenFlow Switches. IEEE journal on selected areas in communications, vol. 38.