Software Defined Network Operating System (SDNOS)

Douglas Comer, Adib Rastegarnia

February, 2017

Introduction

Software Defined Networking (SDN) is an emerging trend for the design of future network management systems. SDN breaks vertical integration by decoupling the control plane from data plane, and provides flexibility that allows software to program the data plane hardware directly. The separation of the control and data planes allows network switches to become simplistic forwarding devices, and allows control logic to be implemented by a logically centralized controller called a Network Operating System (NOS). In recent years, SDN has received attention from service providers, enterprises, and researchers. However, a set of open problems remain unsolved, and must be addressed before the ultimate promise of SDN can be fulfilled.

Problem Statement and Significance

In the current SDN paradigm, an external SDN controller communicates with management applications and with one or more network devices to configure and control the devices. Communication requires that the two communicating parties agree on an Application Program Interfaces (API) to be used. Current SDN controllers employ two APIs that are labeled Northbound and Southbound. The Southbound API defines communication between the controller and a network device. Early SDN work defined the OpenFlow southbound API, and the OpenFlow protocol continues to dominate the southbound protocol space. OpenFlow allows a controller to update flow table rules, and to specify associated actions to be performed for each of the flows that pass through a given network device. The current software-defined management architecture exhibits several weaknesses as follows:

- **Low-level interface.** Current southbound APIs control low-level details. For example, although it provides operations agnostic to any vendor, OpenFlow only handles basic details. Proposed programming systems, such as Frenetic [1], provide high-level abstractions to control a network directly, but require a programmer to map abstractions to OpenFlow rules that associate actions with rules that match fields in a packet. To overcome the weaknesses, a set of new abstractions must be designed that allow programmers to focus on end-to-end application requirements instead of the low-level details that OpenFlow exposes. Consequently, we propose to create abstractions that reflect end-to-end requirements.

- **Inadequate functionality.** A flow-based, match-action abstraction cannot specify complex network functions, such as encryption/decryption functions used for security and deep packet inspection (DPI) used for malware detection. Because such functions require examination or transformation of the payload in a packet, header field matching is insufficient. To combat the weakness we propose to focus on abstractions that go beyond flow-based, match-action rules and support flexible network functions.

- **Low-level programming interface.** Typical SDN controllers [2] provide a simplistic RESTful API which means a programmer must specify low-level the details for each flow, and must parse JSON or XML formats to retrieve network topology, network parameters, and statistics. One of the critical pieces of our work will be the design of a new interface that allows programmers to write network management application without worrying about the low-level details of how the information is transported and stored, and without any need to write code that parses data obtained from a device. In essence, we propose to find ways to remove the drudgery from network programming.
Questions To Be Explored

The following summarizes two critical research questions that we propose to explore:

- What are the appropriate abstractions for describing, programming, monitoring, and reasoning about large networks that support a wide range of applications and services?

- What are the tradeoffs between high-level abstractions that can express goals and low-level abstractions that make device configuration efficient? Do high-level abstractions exist that offer flexibility and, at the same time, allow efficient, automatic translation into an equivalent set of low-level specifications?

Main Aims and Objectives

To overcome weaknesses and achieve the goals outlined earlier, we propose to build a Software Defined Network Operating System (SDNOS). The new system will be able to run management applications and offer support for a variety of southbound APIs.

Management application execution. Our SDNOS will run management applications analogous to the way a conventional operating system runs processes. Just as a conventional process uses services provided by its operating system, an SDNOS application will use services provided by SDNOS.

Dynamic support for arbitrary southbound APIs. A conventional operating system uses device drivers to permit applications to use a device without knowing the details of how the device connects to the computer or the protocol used for communication. Our SDNOS will use the same idea to provide communication between a controller and network devices. Thus, a management application will be able to configure a switch without knowing whether the switch uses OpenFlow or some other API because the driver will take care of such details. Furthermore, we envision SDNOS drivers being loaded dynamically, just like device drivers in a conventional operating system. That is, when a new network device is added, the SDNOS will detect the device and load the appropriate driver automatically (subject to policy restrictions, of course).

Research Methodology

The research will span one year, and will consist of four main thrusts.

- First, we will design a set of abstractions that allow us to capture high-level network policies, a conceptual set of functions that devices will use, and a mapping from the high-level abstractions to the conceptual functions.

- Second, we will build a prototype SDNOS. It will be able to run management apps and dynamically load drivers for southbound APIs.

- Third, we will design and implement an example set of drivers that convert our conceptual functions to specific commands. We propose to start with an OpenFlow driver and also choose a vendor-specific interface.

- Finally, we will implement an example set of basic network management applications, and use them to test the functionality SDNOS and its components.

References


Budget

We request support for a half-time research assistant, and funds to present the work at a conference (travel and registration).

<table>
<thead>
<tr>
<th>Item</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2-time Research Assistant</td>
<td>38,265</td>
</tr>
<tr>
<td>Conference registration and travel</td>
<td>700</td>
</tr>
<tr>
<td>Total</td>
<td>38,965</td>
</tr>
</tbody>
</table>