2016 IEEE 24th International Conference on Network Protocols (ICNP)
Poster Paper

OpenFunction: An Extensible Data Plane Abstraction Protocol for Platform-independent Software-Defined Middleboxes

Chen Tian[†] Alex X. Liu[†‡] Ali Munir[‡] Jie Yang[†]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, China*
[‡]*Department of Computer Science and Engineering, Michigan State University, USA*

Abstract—**We propose OpenFunction, an extensible data plane abstraction protocol for platform-independent software-defined middleboxes. The main challenge is how to abstract packet operations, flow states and event generations with elements. The key decision of OpenFunction is: actions/states/events operations should be defined in a uniform pattern and independent from each other. We implemented a working SDM system including one OpenFunction controller and OpenFunction boxes based on Netmap, DPDK and FPGA to verify OpenFunction abstraction.**

I. Introduction

Data plane abstraction is central to Software-Defined Networking (SDN). Currently SDN data plane abstraction has only been realized for *switches* by OpenFlow [1] and its advanced version P4 [2], but not for *middleboxes*. A data communication network has two types of devices, *switches* and *middleboxes*. Switches (including routers in the broader sense) provide packet forwarding. Middleboxes are used for wide variety of networking and security purposes such as Network Address Translation (NAT), Load Balancing (LB), firewalls (FW), proxies, IPSec gateways (VPN), and network Intrusion Detection/Prevention Systems (IDS/IPS).

A dataplane abstraction for middleboxes is needed to realize the vision of software-define middleboxes (SDM). S-DMs provide network operator the ability to dynamically load/unload various network functions without changing the network hardware, similar to what OpenFlow/P4 provides for the switches. For software-define middleboxes (SDMs), a data plane abstraction should be both *platform independent* and *fully extensible*. Platform independence decouples the data plane function semantics and the underlying hardware that realizes the network function. This allows third-party SDM program's data plane to execute at any SDM boxes (*i.e.*, SDM compliant middleboxes) with same semantics but different performance depending on hardware adequacy. Fully extensible means that any new middlebox functionality can be defined by an SDM program abstraction, which is critical to enable innovation for middleboxes. For example, this enables design of a new packet encryption algorithm for VPN, define a new flow state in the data plane for firewall, or subscribe to an event when a specific condition is trigged for IPS.

In this paper, we propose OpenFunction, an extensible and platform independent data plane abstraction protocol for software-defined middleboxes. OpenFunction architecture consists of a logically centralized *OpenFunction controller* and a number of *OpenFunction boxes* distributed around a network, where every box implements OpenFunction abstraction layer, Fig. 1. OpenFunction abstraction exposes an extensible set of

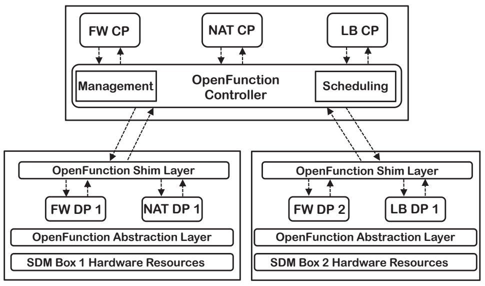

Fig. 1. Software-Defined Middlebox Architecture

elements to the control plane using the modular style. An SDM programmer needs not to be aware of the underlying hardware features of SDM boxes: just define the behaviour of data plane as a data flow graph of processing elements, and focus on the application logic of control plane. Under the hood, OpenFunction defined data plane can be realized by a platform dependent implementation that fully leverages the hardware features of the underlying SDM box. To support new operations, OpenFunction provides a platform-independent pseudo language for specifying customized elements beyond those predefined ones; such a platform-independent pseudo program can be compiled to a platform-dependent element by the underlying box.

II. OpenFunction Based SDM Architecture

In OpenFunction, a network function (such as NAT, LB, or FW) is implemented by a Control Plane (CP) process and a set of Data Plane (DP) processes running on OpenFunction boxes. The CP and DP processes communicate with each other using the OpenFunction protocol. The CP process has a global view (via the controller) and its main role is to manage and deploy the DP processes according to required middlebox functionality. It receives events from DP processes, analyses, makes decisions, and sends commands to DP processes to enforce its decisions. A DP is modeled as a directed acyclic graph where each node is an element that implements middlebox functionality. A DP process has a local view of the network, it receives commands from CP and its main role is to process data plane packets accordingly. Based on the processed traffic, it may generate events for its corresponding CP process. We call a CP process together with its group of DP processes a *Software-Defined Middlebox* (SDM). With OpenFunction, implementing a middlebox functionality on the data plane becomes much simpler as it mostly involves composing a graph of pre-defined elements, possibly with a few user-defined elements. Thus, SDM developers mostly focus on designing CP programs, which are often the most innovative part of their implementation.

OpenFunction data plane abstraction is element oriented, similar to Click. OpenFunction exposes an extensible set of elements to the control plane. An element is a self-contained and functionally independent packet processing unit, such as decreasing the TTL field. The semantics of an element is to take a packet as its input, perform some operations, and either push the packet to the next element or wait for the next element to pull the packet. Each element object is an instance of an element class. Three element classes are actions, states and events. For example, an element may need to update the flow counter state whenever a packet passes through this element; another element may send an event to the control plane. There are two kinds of elements: pre-defined and user-defined. Pre-defined elements are those supported by OpenFunction compliant boxes and user-defined elements are those written by users using a platform-independent pseudo language. We allow multiple DP processes running on the same OpenFunction box, similar to router visualization, where in the same box, different DP processes belongs to different CP processes for different middlebox functions.

The main challenge is how to abstract packet operations, flow states and event generations with elements. A data packet should be abstracted first before any access operation can be defined over it. There are three types of flow states: per-flow, multi-flow, or global. For example, an IDS's data plane might need to record packet counts of each flow, of flows to each destination, and of all the pass-through flows simultaneously. Some operations might depend on the flow states as input: for example, send an event to control plane when the number of packets to a destination exceeds a given threshold. Events can be used in various form according to control plane requirements. Apparently, explicitly defining different states/events for each application scenario separately would result in the exponential growth of the number of elements and unnecessary duplicate implementation.

The key design idea of OpenFunction is that actions/states/events operations are defined in a uniform pattern and independent from each other. Also, it separates three kinds of primitive data plane processing elements: *action*, *state* and *event*. An action element can modify packet data, but can not modify a state or issue an event to control plane; a handful number of state elements are dedicated to update the state records; several event elements are dedicated to issue events. OpenFunction protocol treats every state as a record in a conceptually global key-value store. As a result, states can be accessed in a uniform way. OpenFunction treats the trigger condition of each event as a boolean expression, and the event as a formatted string. As a result, event elements can be abstracted in the *evaluate-format* form.

OpenFunction dataplane faces various other challenges. First, an SDM box may implement different kinds of network functions simultaneously; as a result different elements may interact with each other to complete the service chain. Second, different elements may need to be loaded/unloaded dynamically to support different network functions' requirements. During load/unload, the OpenFunction box must not experience packet drops and hence degrade application performance. We propose a bufferless in-box service chaining based solution to address these challenges. OpenFunction box runs a parent program that creates the memory and defines the data structures for child processes. Each element is then started as a child process that can use the shared memory.

III. Prototype Implementation

We build a proof-of-concept system to verify the OpenFunction abstractio. We develop OpenFunction boxes based on Netmap [3], intel DPDK [4], and NetFPGA [5]. We implement NAT and IPSec gateway as two stateless SDMs on these platforms. The NAT middlebox performs simple address translation; the mapping table is stored in its CP process and the mapping item is send to a DP process when the first packet of a flow comes in. The IPSec gateway uses ESP tunnel model with AES-EBC encryption. The CP negotiates with the remote peer, and sends command to DP processes after security negotiation.

To evaluate, we use a small topology with five DPDK middleboxes and generate TCP flows between source and destination. We initiate flows on path 1 and path 2 which initiates flow rule installation in the switches along the datapath and calculate datapath setup times. Our evaluation shows that it takes less than 100 msec to add a new datapath or service chain in the network. Also, OpenFunction adds less than 20 usec per SDM DP in the service chain.

Lastly, we test the OpenFunction specification performance for the data plane abstraction of each SDM. For each SDM, we use same set of OpenFunction specifications. We test with both small size packets (60 bytes) and large size packets (1400 bytes). Table I shows the throughput performance. For non-stateful SDMs (NAT and IPSEC), DPDK box achieves higher throughput than Netmap-based box in almost all the scenarios. This demonstrates that, even with the same OpenFunction abstraction implementation, the performance actually relies on underlying systems.

TABLE I
Throughput of OpenFunction middleboxes

	Netmap	DPDK	FPGA
NAT Small	0.14 Gbps	1.84	1.64 Gbps
NAT Large	0.20 Gbps	9.96	9.34 Gbps
IPSEC Small	0.02 Gbps	1.2	2.61 Gbps
IPSEC Large	0.17 Gbps	2.8	9.35 Gbps

Acknowledgement

This work is partially supported by the National Science Foundation under Grants CNS-1318563, CNS-1524698, and CNS-1421407, and the National Natural Science Foundation of China under Grants 61472184, 61321491 and 61602194, and the Jiangsu High-level Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] Nick et. al. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, April 2008.
[2] Pat et. al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR*, 2014.
[3] Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX ATC*, 2012.
[4] DPDK Intel. Data plane development kit.
[5] Lockwood et. al. Netfpga–an open platform for gigabit-rate network switching and routing. In *IEEE MSE, 2007.*