FlexForward: Enabling an SDN Manageable Forwarding Engine in Open vSwitch

Rafael D. Vencioneck, Gilmar Vassoler,
Magnos Martinello, Moises R.N. Ribeiro
Federal University of Espírito Santo (UFES)
Vitória - ES - Brazil
{rafael.vencioneck@aluno, gilmarvassoler@ele,
magnos@inf, moises@ele}.ufes.br

Cesar Marcondes
Federal University of São Carlos
São Carlos - SP - Brazil
marcondes@dc.ufscar.br

Abstract—**This paper presents FlexForward, an approach for dynamically manage SDN data plane forwarding mechanisms. This feature leverages SDN flexibility in two aspects: (i) it provides a new management method, in which topological infrastructure requirements are satisfied by the most appropriate forwarding mechanism, residing in each managed network element; (ii) further decoupling between control and data planes in SDN by offering tableless forwarding support. As proof of concept, an implementation with different forwarding methods is carried out in Open vSwitch. Results show that FlexForward is able to achieve seamless switchover between forwarding methods. It also allows efficient tableless forwarding implementations, improving by up to 31% in latency, and 90% in throughput, when compared to regular OpenFlow.**

I. Introduction

The large popularity of Software Defined Networking (SDN) in both academic and industry environments revealed how relevant is to reach network programmability, enabled by decoupling of control from the physical infrastructure. A protocol called OpenFlow (OF) [1] was the first SDN enabler, providing an Open API to program the network gear.

Despite OF improved features in its latest releases, there are yet many practical problems to be addressed. Kong *et al.* [2] points out latency and flow table size as the main OF issues. Latency is increased due to communications between switch and controller, which may take a non-negligible time, and by the use of small flow table sizes, which forces the switch to send more Packet-In messages to controller. Larger flow tables requires more Ternary Content Addressable Memory (TCAM), increasing substantially the switch cost.

These issues must be considered even on OF most promising application domain, the Data Center Networks (DCN), which are divided into network-centric and server-centric designs [3]. While network-centric approach uses a hierarchy cluster of high-end switches with large number of ports to interconnect the servers, server-centric networks approach reduces or eliminates physical switches and routers. This implies that servers work as network devices, doing packet forward besides computational tasks. DCNs typically have thousands of servers and switches [3], generating a large number of flows in each SDN data plane element, possibly causing an unnecessary overload on the remote controller.

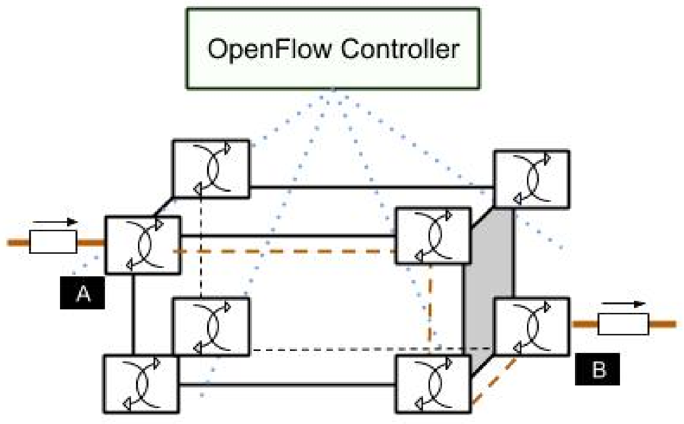
The server-centric approach unfolds new implementation possibilities for network designs and services, such as routing algorithms or traffic engineering. Depending on network design, a specific forwarding model needs to be supported and deployed in the data plane, which must be programmed to understand the necessary modifications for supporting a new algorithm. Although one may argue that this feature could be handled by the SDN central controller, undesired delay and throughput issues may affect the performance of this design.

In this paper we propose FlexForward, an approach to achieve a broader management for network elements by enabling flexible adaptation of forwarding algorithms in server-centric Software Defined DCNs. The topological management, performed by the controller, defines requirements for a given time, and by using its network knowledge, it instructs FlexForward enabled switches to dynamically modify their forwarding methods by sending an OF message. Our implementation is an extension of Open vSwitch (OVS) and its performance is assured by using tableless forwarding techniques.

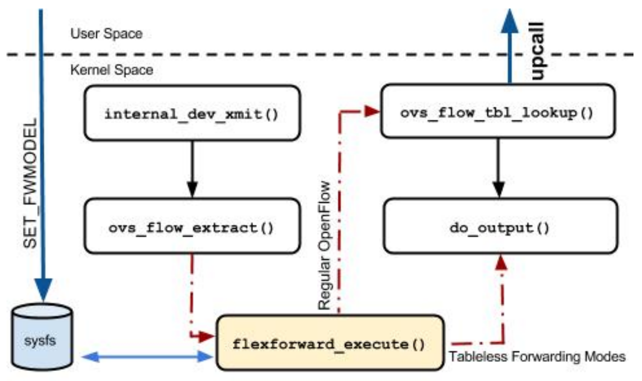
II. FlexForward Overview

Traditional OF enabled equipments provide their table lookup as unique operation for supporting packet switching. For illustration purpose, we assume a HyperCube topology presented at Figure 1(a). In order to configure a route from point A to B, OF controller must insert a rule in each switch between the ends. Packets matching those rules from a table lookup operation are processed by specific actions, to then be forwarded through the path. Often DCN forwarding models, regarding to the data plane, are just simple operations [3]–[5], reaching more efficiency if the switch could itself perform a specific operation for each model. In HyperCube routing algorithm, this is a simple logical XOR operation, who gives the output port for each element.

Based on SDN principles, FlexForward key feature is to provide a flexible forwarding model by offering building blocks to support various models in the data plane. An SDN controller is responsible for managing the whole topology, being able to automatically choose the routing design that suits to a specific network interconnection project. Rather than inserting rules in each switch by sending flow-mod messages that will process packets in a static table lookup model, it only needs to instruct which available method the switch should use by sending OF messages to each of them.



(a)



(b)

Fig. 1: FlexForward: (a) Application in a hypercube topology (b) Overview of switch Implementation

III. Implementation

As proof of concept, FlexForward was implemented as an extension of Open vSwitch (OVS)[1] version 1.9.3, mainly in kernel module. OVS is integrated on Linux kernel 3.3 or greater. Thus, as OVS supports OpenFlow, it became our natural choice for SDN southbound API.

The forwarding methods are placed between the flow extraction and the matching functions, as showed in Figure1(b), making possible to use new headers if necessary, and then forward packets without checking the flow table. In case of regular OF traffic, FlexForward function becomes transparent, not changing usual OVS packet workflow.

It is worth noting that a new forwarding method, when built from scratch, can require thousands of lines of code. For instance, BCube model required 17K lines [3]. Leveraging OVS already implemented forwarding engine, FlexForward extension provides building blocks for easily define a new specific function, which may be implemented between others, reducing implementation cost from thousands to dozens of lines.

[1]http://openvswitch.org

FlexForward improves OpenFlow 1.0 through ”vendor extensions”. New ”vendor actions” were added inside OVS Nicira’s extensions, providing protocol indefinitely expansions without modifying its standard messages. The first message is `SET_FWMODEL`, which tells the switch which forwarding model to use. In order to ensure operation performance, it must keep packet processing in OVS kernel module. Therefore, every variable regarding to network state is stored on *sysfs* linux file system, which allows any information coming from the controller through a vendor message to be handled by OVS kernel module.

FlexForward controller application was developed to run on Ryu[2]. Since Ryu already supports Nicira Extensions, only the new messages was required to be added in it. All controller logic for changing the forwarding model by sending vendor messages, handling OF packet-in (in case of using regular OF mode), and learning the topology by using Link Layer Discovery Protocol (LLDP) was also written.

A. Deployed Forwarding modes

The following forwarding modes were used for a initial deployment.

Regular OpenFlow is the standard mode when FlexForward comes up. It will behave as OF switch until it receives a command to change the forwarding design.

Hypercube topology links switches mainly on server-centric DCNs. The minimal complete configuration is illustrated on Figure 1(a), with eight switches. Each element has three neighbors. To conceive the correct forwarding, the switches must store addresses for themselves and for each of its neighbors with associated port. We use 32 bits variables on *sysfs* to store these values. When a packet arrives, FlexForward extracts the first 32 bit of the destination MAC, which contains HyperCube node’s destination address. A HyperCube node address must have exactly one different bit, when compared with its neighbors addresses, unveiled by a logical XOR operation.

The **KeyFlow** [4] forwarding method is supposed to be used in any topology. It relies on Chinese remainder theorem to design global path ID and switch local keys, in which a MOD operation between the global path ID and switch local key gives the output port. The controller calculates the global path ID and send it inside the packet. The local key is installed on the switch through an OF message, and stored in a 32 bit variable. For validation purposes, the path ID is carried in MAC header destination field.

IV. Evaluation

In order to evaluate FlexForward, we accomplished two main tests using the topology described in Figure 1(a). The first covers overall FlexForward performance through latency, jitter, and throughput for the forwarding mechanisms presented in section III-A. For the second test, we verified packet behavior for a switchover event from HyperCube to KeyFlow. In OpenFlow tests, the controller ran HyperCube algorithm.

[2]http://osrg.github.io/ryu/

Eight virtual machines (VM) were used to set up a minimal HyperCube environment. In order to isolate the traffic, twelve virtual networks were used to emulate point-to-point connections between VMs. The physical machine is an IBM Server System x3200M3, with an Intel Xeon X3430, 2.40Ghz CPU and 32GB of RAM, running VMWare ESXi 5.1.0 [3]. Each VM ran a 32 bit linux Debian 7, kernel 3.2 and 4GB of RAM. A separated VM was used as controller. Figure 1(a) also illustrates the path used to send packets, i.e., from point A to B, which reaches the highest number of hops for that topology.

A. Performance Analysis

FlexForward performance analysis starts with network latency comparison between available forwarding mechanisms, accomplished by 10000 uniformly sent ping packets to obtain Round-trip delay time (RTT), with 1 millisecond interval between them. For OpenFlow, the rules were installed with infinite duration, meaning no packet-in after the first switch requisition to controller.

As results, which is showed in Figure 2(a), around 90% of HyperCube forwarding mechanism packets had RTT below 0.25 ms while KeyFlow presented 80%, followed by OpenFlow, with only 15%. OpenFlow results were improved because of OVS' flow-caching feature that decreases the average time for table lookup of replicated flows, as equal pings. Nevertheless, OpenFlow average latency was 31.5% higher than HyperCube and around 25% higher than KeyFlow.

The second analyzed variable was jitter, a latency variation for arrived packets. Low jitter is a critical network parameter for some services like VoIP and other real-time transmissions. To measure it, iperf[4] was used, collecting information every 0.5 sec. Figure 2(b) shows that while 99% of jitter values were lower than 0.05 ms for both HyperCube and KeyFlow, OpenFlow had 50% greater than 0.3 ms.

Lastly for performance, each forwarding mechanism had its throughput compared, as showed in Figure 2(c). The server has 1G physical interfaces, but FlexForward tests were not limited to it, because communication between virtual interfaces achieved the maximum value that CPU and bus could handle. Once more, flows were installed without timeout for OF. For better graph presentation, instant variations were eliminated by using simple rolling average. Figure 2(c) shows HyperCube and KeyFlow performing twice better than OpenFlow. In average, HyperCube reaches 1.6 Gbit/s, followed by KeyFlow with 1.55 Gbit/s, while OpenFlow only achieves 0.84 Gbit/s.

Results confirmed that storing less network state improves switch forwarding speed for server-centric networks. Likewise, even with pro-actively installed flows, eliminating flow table lookup showed better latency, throughput and jitter values for HyperCube and KeyFlow. FlexForward provides faster and uniform SDN data plane for DCNs.

B. Forwarding Method Switchover Analysis

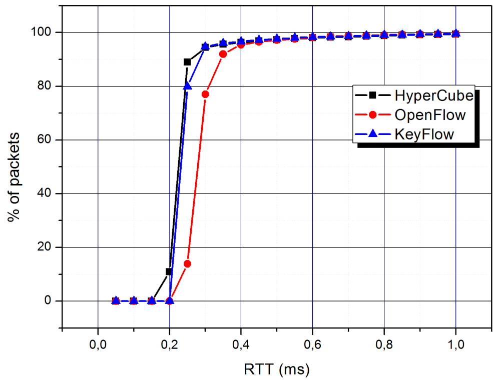
In order to analyze forwarding mode switchover impact on the network, UDP packet bursts were generated by iperf, with different speed rates, represented by each line in the graph. The change, from HyperCube to KeyFlow, occurs at 20 seconds. In our tests, iperf could not generate rates greater than 800 Mbit/sec for UDP.

As showed in Figure 3, a small degradation, during less than 0.5 second, occurs at the swap event. That is due to the reconfiguration of switches, as packets already in the path generated for the old forwarding method are consequently lost.

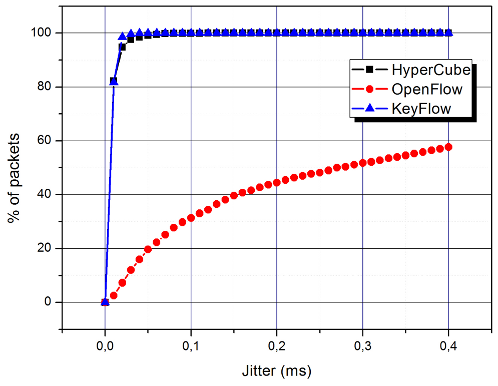
These results allows network managers to implement algo-

[3]www.vmware.com/br/products/esxi-and-esx/overview

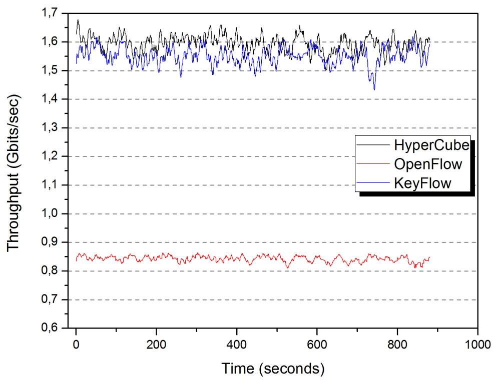
[4]http://iperf.fr/



(a)



(b)



(c)

Fig. 2: FlexForward Performance: (a) Latency, (b) Jitter, and (c) Throughput comparison among OpenFlow, HyperCube, and KeyFlow forwarding mechanisms

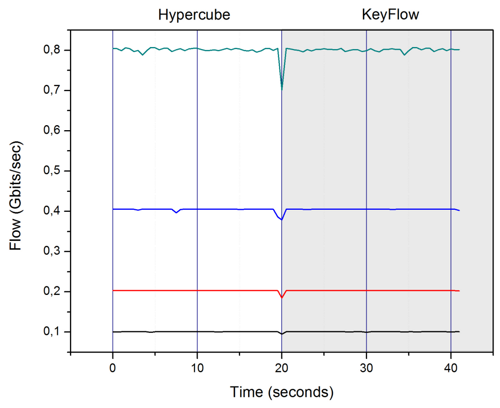


Fig. 3: FlexForward end-to-end switchover through different bandwidth occupation

rithms in accordance with DCN designs that need multiple customized packet forwarding engines, without introducing new Layer 2.5 headers. For instance, in case of failures breaking the hypercube properties, then the routing algorithm can be changed by the controller, by promptly sending a command to all network elements.

V. Related Work

There are many works related to flexible data plane hardware implementations. ServerSwitch [6] is a fully programmable commodity server with customized network card and driver for server-centric architectures. CAFE project [7] provides simple APIs for controlling switch behavior on packet forwarding. SwitchBlade [8] permits custom forwarding methods to be executed in parallel by referencing to each data plane as a Virtual Data Plane(VDP), providing to each VDP a separate processing pipeline. Each forwarding model is loaded as a module.

Recently, Bosshart et al. [9] has proposed an architecture for switching chips, called Reconfigurable Match Table (RMT), that allows changing the data plane forwarding behavior without coding the hardware. Nevertheless, some forwarding models have their design impaired if every packet without an entry in the table is sent to the controller, potentially causing degradation in latency. Also, the network state needed to be stored is still considerable.

FlexForward, differs from the previous works by leveraging the best of SDN and server-centric DCN, providing flexible forwarding at the data plane relying in a software design in kernel space to enable the switch forwarding setup. It uses the SDN control plane allowing the controller to decide which forwarding model is better for a given situation, reconfiguring the network elements to the most suitable technique.

VI. Conclusions and Further Work

This paper introduced FlexForward as a proposal to help achieving a new management model with more decoupling and flexibility on Data Center software defined architecture. It works by enabling easier implementation of new forwarding methods and allowing greater management on the network element. The controller acts as a manager, leveraging its centralized network knowledge in order to automatically instruct a forwarding method switchover to the switches.

It is implemented by modifying OVS to understand new OF vendor messages, keeping it fully compatible with the protocol, but also talking customized messages. In addition to enabling a way for designing new efficient forwarding methods on the same switch, FlexForward provides support for a seamless switchover between them.

In our tests, FlexForward presented two times better throughput performance, in comparison with OpenFlow. Also, RTT and jitter values were 90% below 0.25 ms and 99% below 0.05 ms, versus OF having 15% below 0.25 ms and 50% above 0.3 ms, respectively.

As future work, we plan to implement FlexForward considering larger DCN topologies, with more forwarding options, as well as providing multiple modes in parallel, creating network virtualization. Another direction to be explored will be the use of hardware instruction sets provided by some Intel chips, using for example Intel DPDK[5].

References

[1] McKeown, Nick et al., “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[2] X. Kong, Z. Wang, X. Shi, X. Yin, and D. Li, “Performance evaluation of software-defined networking with real-life isp traffic,” in *Computers and Communications (ISCC), 2013 IEEE Symposium on*, July 2013, pp. 000 541–000 547.

[3] C. Guo et al., “Bcube: A high performance, server-centric network architecture for modular data centers,” ser. SIGCOMM ’09. New York, NY, USA: ACM, 2009, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592577

[4] M. Martinello, M. Ribeiro, R. de Oliveira, and R. de Angelis Vitoi, “Keyflow: a prototype for evolving sdn toward core network fabrics,” *Network, IEEE*, vol. 28, no. 2, pp. 12–19, March 2014.

[5] C. R. Ramon Ramos, Magnos Martinello, “Slickflow: Resilient source routing in data center networks unlocked by openflow,” in *Proc. of IEEE Local Computer Networks 2013 Conference*, 2013.

[6] Guohan Lu et al., “Serverswitch: A programmable and high performance platform for data center networks,” in *Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’11. Berkeley, CA, USA: USENIX Association, 2011, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972460

[7] G. Lu, Y. Shi, C. Guo, and Y. Zhang, “Cafe: A configurable packet forwarding engine for data center networks,” in *Proc. of the 2Nd ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO ’09. New York, NY, USA: ACM, 2009, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/1592631.1592638

[8] M. B. Anwer, M. Motiwala, M. b. Tariq, and N. Feamster, “Switchblade: A platform for rapid deployment of network protocols on programmable hardware,” in *Proc. of the ACM SIGCOMM, 2010*, ser. SIGCOMM ’10. New York, NY, USA: ACM, 2010, pp. 183–194. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851206

[9] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, “Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn,” in *Proc. of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM ’13. New York, NY, USA: ACM, 2013, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486011

[5]http://www.intel.com/go/dpdk