Content Distribution and OpenFlow: a Reality Check

Ahmed Amine Loukili*, Damien Saucez*, Thierry Turletti*, Mathieu Bouet†
∗ Inria Sophia Antipolis – France
† Thales – France

***Index Terms*—SDN, benchmark, content distribution**

***Abstract*—With the advent of virtualization and network function softwarization, the networking world shifts to Software Defined Networking (SDN). The OpenFlow protocol is one of the most suitable candidates to implement the SDN concept. In the meanwhile, the generalization of broadband Internet (mobile, cable, DSL, fiber etc.) has led to massive content consumption. However, while content is usually retrieved via layer 7 protocols, OpenFlow operations are performed at lower layers (layer 4 or lower) making the protocol completely ineffective to deal with content. To address this issue, we proposed and developed an API to manage content in OpenFlow networks. We implemented this API using open source software and study the impact of logical centralization suggested by SDN on network performances.**

I. Introduction

Over the last decade we have seen the generalization of virtualization in networking. More specifically, the SDN approach aims at making networks programmable thus enabling network operators to automate their network management. OpenFlow is one of the most used SDN protocol. It relies on a separation between the control plane and the data plane [1] and match-action rules on layers 1 to 4. In parallel to the advent of SDN, network communications have diverted to massive content distribution via the HTTP protocol that operates at the layer 7 of the networking stack (e.g., YouTube). As a result, OpenFlow cannot be directly used to manage content distribution.

To reconcile content distribution and OpenFlow, we developed a content distribution RESTful API and implemented it by extending the OpenFlow controller and using HTTP proxy middleboxes. The proxies determine the contents that are actually requested within TCP flows. This information is then propagated to the controller that can make the association between contents and TCP flows to finally optimize the network via OpenFlow rules.

With the demonstration, we show that the choice of the architecture implementing the API has a major effect on the control plane and therefore on the feasibility of the solution: by introducing the notion of control plane caches, a centralized approach can be used to manage content distribution with OpenFlow without impairing network performances.

In Sec. II we present the architecture that implements our content distribution API relying on OpenFlow. In Sec. III we provide first performance figures for the architecture. Finally, in Sec. IV we conclude this work.

II. Architecture

Our architecture provides content distribution management with OpenFlow. To that aim, we developed a RESTful API[1] modeling the content distribution network as an abstract graph. The data model defines three atomic types: *compute* nodes for functions related to processing (e.g., process, CPU), *storage* nodes for data related operations (e.g., file, hard-drive), and *network* nodes for all operations related to the network (e.g., routes, NIC) and the composition of complex entities is called a *slice*. For example, a router is composed of network ports (network nodes), processors (compute nodes), and memory (storage nodes). All these components are modeled independently and grouped by a slice to compose the router.

The API links contents and network but content distribution relies on HTTP while OpenFlow matching is limited to the lowest layers of the network stack (i.e., up to transport layer). Thus, implementing the API requires to use HTTP proxies. The interaction between OpenFlow, HTTP proxies, and the centralized controller is depicted in Fig. 1 where all components interact by the means of the API.

The API links contents and network but content distribution relies on HTTP while OpenFlow matching is limited to the lowest layers of the network stack (i.e., up to transport layer). Thus, implementing the API requires to use HTTP proxies. The interaction between OpenFlow, HTTP proxies, and the centralized controller is depicted in Fig. 1 where all components interact by the means of the API. We leverage the logically centralized approach of OpenFlow to simplify the management of the infrastructure. However, networks being inherently distributed systems, and concentrating all information and decisions would impair performances by causing high signaling load. Therefore, as illustrated by Fig. 1, decisions are performed centrally by the controller, but pushed in the network components that dispose of a control plane cache, thus avoiding to load the controller with data plane events.

The central entity controlling the network extends the OpenFlow controller to link content and network operations with the API. This API is used to retrieve content information from the HTTP proxies and to manage them, but also to program the storage or migration of contents on various storage servers of the network. Furthermore, combined with the OpenFlow controller, it permits to manage OpenFlow switches to optimize network operations as well.

[1] Available at https://github.com/Rzozi/Contentdistribution

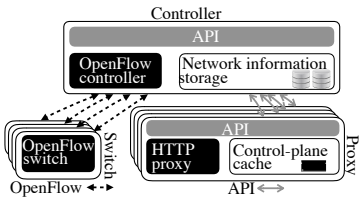
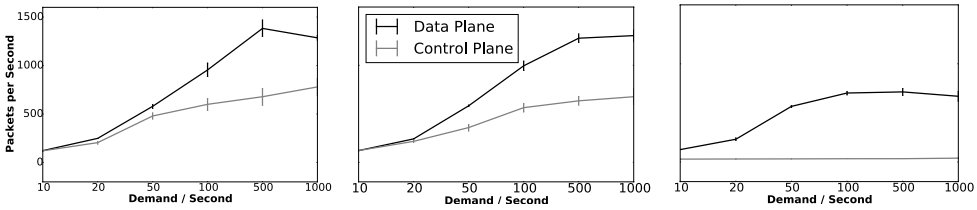


Fig. 1. SDN-enabled content distribution architecture.



(a) fully-centralized (b) semi-centralized (c) fully-decentralized

Fig. 2. Evolution of the control and data plane rates with content demand.

A naive way of implementing the centralized control would be to ensure that at any time the central controller disposes of all information of the network and that all the components of the network query the controller whenever they have to make a decision. This approach guarantees the decision consistency but at the expense of the scalability since the control plane load would be directly proportional to the data plane load.

To tackle this issue while keeping the advantage of centralization, we extend the concept of OpenFlow flow table. We propose *control plane caches* that store, locally on the network components, the result of computations of the controller. This way, instead of querying the controller to know what operation to perform, network components query their local control plane cache. In OpenFlow switches, the control plane cache corresponds to the flow table, while in the HTTP proxies it is an indirection table that determines how to tag packets so that they can be processed adequately by OpenFlow switches. With this approach, managing the control plane becomes a database synchronization problem where network components feed the database with data plane information and the controller process the information to decide data plane operations.

The controller bases its decision on knowledge of the network status and traffic flows. However, in most situations, the controller is out-of-band and must query the data plane components (i.e., switches and proxies) to obtain the information. Therefore, to avoid the data plane components to send such information to the controller at the data plane rate, the control plane cache is also used to temporary store data-plane statistics collected by the data plane components and the controller periodically retrieves this information. The frequency of updates is a tradeoff between the scalability and the reactivity of the system.[2]

III. Demonstration

We prototyped our architecture with open source software. We use Open vSwitch [2] and Floodlight [3] for OpenFlow parts, and CherryProxy for the HTTP parts [4]. The REST API and the control plane cache are implemented with Flask [5] and PostgreSQL [6].

For this lecture demonstration, we will show the impact of the control plane cache on the network load. To that aim, we consider three scenarios: (i) fully-centralized, (ii) semi-centralized, and (iii) fully-decenralized control plane caches. The first scenario is the case where all information and decisions are treated by the controller, i.e., no cache. The semi-centralized scenario is the situations where the control plane caches only store the data plane decision made by the controller. Finally, the fully-decentralized control plane scenario is the one where all information are cached in the data plane network components.

Fig. 2 is an example of results we will show during the demonstration. It shows the evolution of the data plane and control plane rates with the content demand rate. Each network function runs in a separate virtual machine deployed on the same host. As expected, the fully-decentralized approach guarantees a control plane load independent of the data plane rate. On the contrary, the fully- and semi- centralized scenarios show a rather linear correlation between control and data plane rates. Interestingly, when control plane caches are used, the proxy requires more CPU cycles to process flows, which reduces the data plane rate. With a fully decentralized control plane cache this phenomenon is compensated by the suppression of traffic with the controller but when the controller is involved, it results that a fully centralized approach offers slightly better, though not significant, data plane rates.

IV. Take away messages

We present an architecture to implement a content distribution API in OpenFlow networks. Our architecture demonstrates that OpenFlow can be used to distribute content using a centralized approach without impairing network performances.

Biographies

Ahmed Amine Loukili received his master degree in Ubiquitous networking and computing (UBINET) from the Universit'e Nice Sophia Antipolis, France, and an engineering degree in Telecommunication and networking from the National School of Applied Sciences of Tangier (ENSAT), Morocco in 2016. His master internship at Inria Sophia Antipolis is entitled Content Distribution over SDN. His current research interest is NFV/SDN.

Damien Saucez received his master degree in Computer Science engineering from Université catholique de Louvain in 2007 and his Ph.D. thesis entitled Mechanisms for Interdomain Traffic Engineering with LISP in 2011. Since 2011, he has been working as a resercher at Inria Sophia Antipolis. His current research interests are information-centric networking (ICN) (e.g., routing and congestion control), Software Defined

[2]Upon exceptional events, the control plane cache can be bypassed to directly inform the controller.

Networking (SDN) (e.g., resiliency and robustness), and large-scale experimentations. He is actively contributing to the IETF and IRTF.

Thierry Turletti is a senior research scientist in the DIANA team at Inria Sophia Antipolis. He received M.Sc. and Ph.D. degrees in computer science from the University of Nice-Sophia Antipolis, France. His current research interests include Software Defined Networking, network experimentation platforms, and wireless networking. He is serving on the editorial boards of the Wireless Networks and Advances in Multimedia journals.

Mathieu Bouet received his PhD degree from the Computer Science laboratory (LIP6) of Pierre & Marie Curie University (UPMC), France, in 2009. Since 2009, he has been working as a research engineer at Thales Communications & Security, where he first conducted research on autonomic networks and cyber-security and then initiated and developed activities on software-defined networking and network functions virtualization. In 2014, he became the technical manager of the Network Softwarization research group. He coordinates several collaborative research projects on this topic. His current research interests are the softwarization of IP and mobile networks (routing and traffic engineering, mobility, new architectures and performance).

ACKNOWLEDGMENTS

This work was supported by the ANR DISCO Project (ANR-13-INFR-013).

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "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] "Open vSwitch," see http://openvswitch.org.

[3] "Floodlight OpenFlow Controller - Project Floodlight," see http://www.projectfloodlight.org/floodlight.

[4] "CherryProxy - a filtering HTTP proxy extensible in Python | Decalage," see http://decalage.info/python/cherryproxy.

[5] "Flask (A Python Microframework)," see http://flask.pocoo.org.

[6] "PostgreSQL: The world's most advanced open source database," see https://www.postgresql.org.