Seamless content distribution with OpenFlow

Ahmed Loukili, Damien Saucez, Thierry Turletti, Mathieu Bouet

Seamless content distribution with OpenFlow

Ahmed Amine Loukili*, Damien Saucez*, Thierry Turletti*, Mathieu Bouet†
* Université Côte d'Azur, Inria – France
† Thales – France

ABSTRACT

With the advent of virtualization and network function softwarization, the networking world shifts to Software Defined Networking (SDN) and OpenFlow is one of the most suitable candidates to implement the southbound API. In the meanwhile, the generalization of broadband Internet 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 ineffective to deal with contents. To address this issue, we define an abstraction to unify network level and content level operations and present a straw-man logically centralized architecture proposal to support it. Our implementation demonstrates the feasibility of the solution and its advantage over fully centralized approach.

1. INTRODUCTION

Over the last decade we have seen the emergence of Software Defined Networking (SDN) making networks programmable thus enabling network operators to automate their network management. 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). However, OpenFlow ([8]) that is the most prominent solution to implement SDN concepts relies on match-action rules on layers up to 4 such that it cannot be directly used to manage content distribution.

Several approach are taken to reconcile content distribution and SDN ([6, 7, 9]) and our approach is to define an *abstraction* that unifies the network and content related tasks. Our abstraction uses network and content operations as primitives and leaves the details of which layers the operations must be performed on to the implementation of the abstraction itself. To implement it, we propose an *architecture* offering a centralized control with a distributed state by the means of *control plane caches*. To validate our straw-man proposal, we implemented a proof-of-concept leveraging, OpenFlow controllers, HTTP proxy, and a REST API. This implementation shows that by introducing the notion of control plane caches, a centralized approach can be used to manage content distribution with OpenFlow.

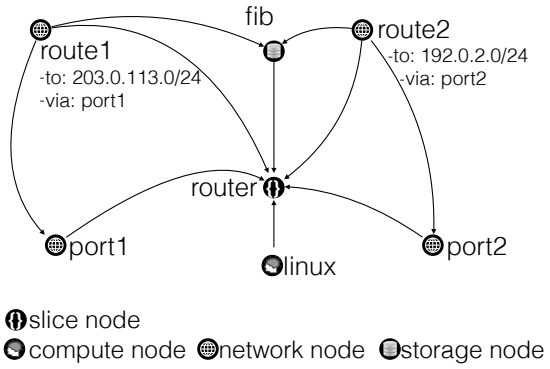


Figure 1: Model of a two ports router with two routes.

In the remaining of the paper, Sec. 2 defines our OpenFlow content distribution abstraction, Sec. 3 presents a straw-man logically centralized architecture proposal to implement this abstraction, and Sec. 4 provides a first evaluation of the architecture based on a prototype implementation. Finally, Sec. 5 concludes this work.

2. NETWORK ABSTRACTION

To leverage the SDN principle and distribute contents using OpenFlow, it is necessary to provide a new layer of abstraction that hides the notion of network layers to focus on network operations and on contents. To that aim, we propose to model the content distribution network as an abstract directed graph where each node represents an entity or an operation of the network and where edges indicate the relationship between them.

Our model defines the following three atomic types of node: (*i*) *compute*, (*ii*) *network*, and (*iii*) *storage*. Compute nodes are used to represent entities and functions related to processing (e.g., CPU, process) while network nodes are used to represent entities and functions related to networking (e.g., NIC, routes). Finally, storage nodes are used to represent entities and operations related to contents (e.g., hard-drive, file). As elements

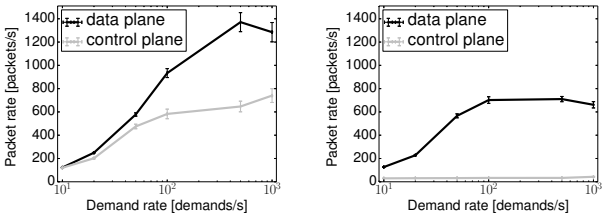
of a content distribution network are often constituted of compute, network, and processing parts our model defines the meta-node type called *slice* that allows to compose complex entities. Each node is allocated a unique identifier such that, in conjunctions with types and attributes, the validity of the model can be verified with static analysis before deploying it in the network.

Fig. 1 shows a simple two-network-ports Linux router with two routes abstract graph representation. The router is composed of two network ports (`port1` and `port2`) modeled as network nodes, the Linux instance is modeled as a compute node, and the FIB modeled as a storage node. Each route (`route1` and `route2`) is represented by a network node with attributes corresponding to the destination and the port to use. All these independent nodes are linked to a slice representing the whole router.

3. ARCHITECTURE

Our abstraction puts at the same level content and network features but content distribution usually relies on HTTP while OpenFlow matching is limited to the lowest level of the network stack. Unfortunately, HTTP is transported over TCP and by virtue of separation principle, no content information is visible at the TCP layer that is the upper layer at which OpenFlow works. In addition, to offer reliability, TCP relies on sessions which state is built before the HTTP flow itself. To implement the abstraction, it is thus necessary to use HTTP proxies that will terminate the TCP connection until they learn the content actually requested in order to reconstruct the TCP sessions to the endpoints fulfilling the policies defined in the model. As proxies operations are computationally expensive, they are only performed at the edge where HTTP flows enter the network. Each such proxy is also an OpenFlow switch and flows are colored by the proxy with tags that can be processed directly by OpenFlow. To ensure core stability, it is possible to predefine the tags, at most one per potential path, such that the arrival of a new flow does not cause flow table modifications in the core network.

We leverage the centralized nature of OpenFlow to simplify the abstraction implementation via the OpenFlow controller. However, networks being inherently distributed, concentrating all information and decisions would impair performances by causing high signaling load. To alleviate the signaling cost but keep the centralized decision making, we introduce the concept of *control plane cache*. The principle of control plane caches is the same as the flow tables in OpenFlow but extended to our abstraction needs such that packet tagging and statistics are always performed locally by using the control plane cache. Before taking specific decisions, the controller gets statistics from the control plane caches and updates them with the new tagging rules. To ensure the scalability of the overall system, the abstraction is implemented by the intermediate of a REST API.



(a) Centralized (b) Control plane caches

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

4. EVALUATION

We prototyped our proposition with open source software. We use Open vSwitch [4] and Floodlight [3] for OpenFlow parts, and CherryProxy for the HTTP parts [1]. The REST API and the control plane cache are implemented with Flask [2] and PostgreSQL [5].

The implementation demonstrates the feasibility of the model and the architecture and illustrates the benefits of the control-plane caches. The first scenario is the case where all information and decisions are treated centrally by the controller, i.e., without cache. The second scenario corresponds to the case where both tagging and statistics are ensured by the control plane caches.

Fig. 2 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. The network is fed with HTTP requests sent according to a Poisson process which average is given on the x-axis. As expected, the control-plane caches approach guarantees a control plane load independent of the data plane rate. On the contrary, the centralized scenario shows a direct 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 data plane rates. This performance drop can be explained by our suboptimal implementation of the control plane with SQLAlchemy that adds overhead.

5. CONCLUSION

This work proposes an abstraction and a straw-man architecture with logically centralized decision making to implement content distribution with OpenFlow. Our first prototype demonstrates the feasibility of the approach and its potential performance gains.

Acknowledgments

This work is funded by the French ANR under the "ANR- 13-INFR-013" DISCO project on SDN.

6. REFERENCES

[1] CherryProxy - a filtering HTTP proxy extensible in Python | Decalage. See `http://decalage.info/python/cherryproxy`.

[2] Flask (A Python Microframework). See `http://flask.pocoo.org`.

[3] Floodlight OpenFlow Controller - Project Floodlight. See `http://www.projectfloodlight.org/floodlight`.

[4] Open vSwitch. See `http://openvswitch.org`.

[5] PostgreSQL: The world's most advanced open source database. See `https://www.postgresql.org`.

[6] A. Chanda and C. Westphal. Contentflow: Mapping content to flows in software defined networks. *CoRR*, abs/1302.1493, 2013.

[7] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker. Less pain, most of the gain: Incrementally deployable icn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 147–158, New York, NY, USA, 2013. ACM.

[8] 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.*, 38(2):69–74, Mar. 2008.

[9] Y. Sakurauchi, R. McGeer, and H. Takada. Open web: Seamless proxy interconnection at the switching layer. In *Networking and Computing (ICNC), 2010 First International Conference on*, pages 285–289, Nov 2010.