From Software Defined Network To Network Defined for Software

Celio Trois
Departamento de Informatica
Universidade Federal do Parana (UFPR)
Curitiba - Brazil
ctrois@inf.ufpr.br

Magnos Martinello
Departamento de Informatica
Universidade Federal do Espirito Santo (UFES)
Espirito Santo - Brazil
magnos@inf.ufes.br

Luis C. E. de Bona
Departamento de Informatica
Universidade Federal do Parana (UFPR)
Curitiba - Brazil
bona@inf.ufpr.br

Marcos D. Del Fabro
Departamento de Informatica
Universidade Federal do Parana (UFPR)
Curitiba - Brazil
didonet@inf.ufpr.br

ABSTRACT

Nowadays, Big Data applications exchange huge amounts of data, highly demanding network guarantees for bandwidth and low latency. However, network equipments did not provide a standard interface to control dynamically the resources. Software-Defined Network (SDN) has emerged to support network programmability, but it provides a programming model devoted to network operators. This paper presents the Network Overlay Framework (NoF), which enables networks to be defined based on application requirements. NoF provides a programming language which allows the application specialists to program the network. A compiler translates NoF programs into instructions according to the underlying network technology. To prove the effectiveness of NoF, programs were implemented to express bandwidth guarantees on Hadoop traffic and also to route Hadoop flows through overlay networks.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations; D.3.4 [**Programming Languages**]: Processors—*Code generation, Compilers*

Keywords

Software-Defined Networking, Network Overlay, Big Data

SAC'15 April 13-17, 2015, Salamanca, Spain.

ACM 978-1-4503-3196-8/15/04...$15.00.
http://dx.doi.org/10.1145/2695664.2696011

1. INTRODUCTION

Big Data processing takes advantages of cloud data centers to process large-scale data with high efficiency. Cloud networks have particular requirements, such as end-to-end quality of service and control over routing and latency. However, current network stack provides a best-effort service model for hosts communication, which is not enough to satisfy the on-demand data access requirements [5].

Software-Defined Networking (SDN) introduces new possibilities for network management and configuration. There are different studies proposing high-level SDN programming languages, but these works focus mostly on network policies [3], providing higher-level abstractions targeted to network operators. There are also works using SDN to optimize the network for specific applications, by managing virtual topologies and providing end-to-end communication [4, 7] and by affording quality of service [2].

Unlike network operators, application specialists are, usually, not familiar with network definitions and concepts. These professionals know exactly when the application will exchange information, and which requirements are needed for each communication phase. On one hand, there are high-level languages, but they still require advanced networking knowledge. On the other hand, there are punctual studies to solve problems of specific applications. However, there is a gap between these proposals to allow the specialists to express their modifications on the network.

To overcome this gap, we present the Network Overlay Framework (NoF). NoF was designed to provide high-level abstractions, defining network according to the applications on top of it. The abstractions include the (i) provisioning and controlling virtual overlay network topologies. (ii) To ensure bandwidth guarantees, (iii) to allow the scheduling of the date/time and for how long the rules will affect the network, and (iv) to exempt the need to know in which hosts the application is being executed. These abstractions were designed modularly to be activated on demand by application specialists, in order to control the network resources at runtime and according to the application needs. For NoF design, we assume that no modification is required on existing

user applications, and information from hosts and processes can be used to trigger events to change the network state.

As a proof of concept, a set of NoF modules where implemented to control a Hadoop cloud. A first experiment demonstrated the expressiveness of NoF by managing Hadoop flows through overlapped topologies. In this experiment, there was a reduction in Hadoop jobs execution time above 27%. Other experiments were conducted to validate bandwidth guarantees and application-based operations to ensure quality of service (QoS). These experiments showed a reduction in execution time varying between 18% and 66%.

This paper is organized as follow: Section 2 introduces the motivation for this work. Section 3 presents the Network Overlay Framework architecture, describing its language, compiler and configuration engine. The experiments and results are reported in Section 4 and finally, Section 5 presents the conclusion.

2. MOTIVATION

This work was motivated by the lack of artifacts that allow specialists to define the network characteristics according to application requirements. The existing works follow two different approaches: raise the level of SDN programming languages and tune the network for one specific application.

SDN has emerged to enable the network programmability. Some controllers offer an API to enable the applications to modify the network. Floodlight, POX, and Beacon are examples of controllers that provide these APIs. There also exist different higher-level SDN languages such as Frenetic, FML and Procera [3].

SFNet [9] introduces an API allowing to query the congestion state of the network and providing services such as bandwidth reservation and multicast. PANE [1] provides an SDN controller and an API to request network resources, query the state of the network, or provide hints about future traffic patterns. Merlin [6] framework implements logical topologies and a high-level language that generates instructions for different switches and controllers.

All these approaches allow applications to modify the network behavior. But they require advanced knowledge in network operations, not providing adequate abstractions to application specialists. Furthermore, the applications must be modified to call the specific APIs to adjust the network.

There are also works focusing on network optimization for specific applications. Wang et al. [7] studied Big Data traffic patterns and proposed network configurations to optimize Hadoop job scheduling by modifying the topology and routing configuration mechanisms. Monga [4] proposed the use of a virtual switch to provide end-to-end virtual circuit.

An API for QoS management in real-time interactive applications (multiplayer online games and simulation-based e-learning) was implemented in [2]. Some network requirements were specified: update QoS in a flexible way; specify different network requirements for different data flows; should allow QoS on data flows; and should be able to place timing-based requirements on the network.

The development of NoF was motivated by two main factors not completely addressed in previous works: (i) the absence of languages that can be used directly by application specialists, and (ii) there exist few studies to improve the application performance by programming the network.

3. NETWORK OVERLAY FRAMEWORK

Network Overlay Framework (NoF) is composed by a programming language, a compiler and a mechanism to configure the equipments. Figure 1 presents an overview of NoF architecture, showing the interaction between these modules. The NoF language provides high-level building blocks to control network topologies, forwarding paths and bandwidth guarantees.

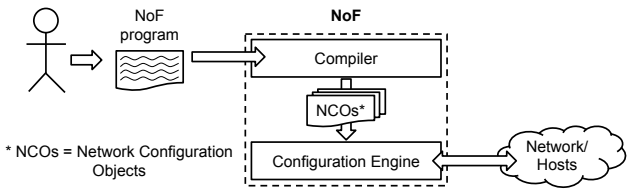

Figure 1: NoF architecture overview.

The Compiler transforms NoF programs in Network Configuration Objects (NCOs). These objects contain instructions to configure the network. The module called Configuration Engine is responsible for receiving the objects generated by NoF Compiler and installing the NCOs' rules on network equipments.

NoF Programming Language

NoF presents a high-level language composed by sets of operations and services. The keywords used in the language attempt to get closer to the nomenclature used by application specialists.

Operations were divided into three groups: (i) Matching operations can be done on traditional network fields (MAC address, IP address, TCP port), but also on host information such as hostname, system load, bandwidth usage or the names of processes wherein new network connections will be monitored. (ii) Timing operations allow services to be scheduled; these operations specify when (date/time) the configurations will be installed and for how long they will act on the network. (iii) Query operations provide abstractions to read network state, for example, link state, transmission errors and bandwidth usage.

Services define the modifications that will be implemented on the network. We have implemented a prototype of NoF to attend the requirements presented in Section 2. These services ensure bandwidth guarantee, manage overlay networks, and flow based forwarding. New services can be added to attend other requirements such as latency or multicasting.

Figure 2 presents some examples of NoF programs. Figure 2.a shows one example where all hosts are monitored. When the system load of any host gets higher than 90% so, the new connections to process *apache*2 receive low priority on QoS queues, and the HTTP traffic is redirected to other instances of *apache*2. In the second example (Figure 2.b) all hosts monitor new connections to processes *hadoop* and *hdfs*. When new connections are received, NoF redirects the flows of these connections through the specified virtual topology.

Compiler

The Compiler is responsible for receiving NoF programs, compiling and generating the NCOs. The Compiler is com-

```
a) NoF : {
   operations(process_name = ("apache2") &&
      system_load =("> 90")),
   services(qos = ("LOW"));
}
```

```
b) NoF : {
   operations(process_name = ("hadoop", "hdfs")),
   services(virt_topo = ("spanning_tree_1"));
}
```

Figure 2: Examples of NoF programs

posed by four main modules: Intent Manager, Topology Manager, Network Abstraction and Host Information.

Intent Manager is responsible for managing the abstractions for different SDN languages or controllers APIs. This module implements a generic interface and uses design patterns with adapters to allow NoF to be extended with new components. The adapters contain skeleton codes to translate the NoF language *services* into specific controllers code.

Topology Manager maintains all network topology information, including switches, links and hosts. This module is responsible for identifing which equipment shall be configured. For example, to perform QoS between two hosts, Topology Manager identifies all equipements involved in communication, applies the right fields in the skeleton received from Intent Manager, and finally, generates the NCOs.

The Network Abstraction Module is responsible for managing virtual network topologies and also for implementing flow delivery mechanisms to provide end-to-end communication. When new virtual topologies are instantiated, this module keeps track of which flows has to be redirected to which virtual port, allowing different matches to be forwarded through different virtual topologies. This module also generates NCOs to collect information from hosts.

Configuration Engine

The Configuration Engine is in charge of receiving and executing the configurations contained in the NCOs. This module was implemented as a client/server architecture. One host runs the server that is called Coordinator process, and all other hosts run the Client processes. This paper focuses only demonstrating the operation of NOF. The overhead imposed by architecture, as well as performance measures, will be presented in future work.

The Coordinator process is responsible for receiving the NCOs and applying their configuration commands on network equipments. Both Coordinator and Client processes implement locally an array of NCOs.

Clients only receive NCOs generated by the Host Information Module, whose are specific to get host information (NCO_h). The NCO_h specifies a command line to be executed in the host console. The command line is executed periodically, and if the result fits the specified match, the Client generates a new NoF program to apply the *services* specified in NCO_h. This program is compiled, and the new NCOs are sent to the Coordinator process.

4. EVALUATION

Experimental results were conducted using Hadoop as data-intensive computing platform. Hadoop relies on a distributed filesystem called HDFS to store and manage data in the cloud. The evaluation methodology was structured in two parts. The first part consisted of analysing the impact on jobs execution time when redirecting Hadoop flows through overlay networks controlled by NoF. The second part was devoted to study the bandwidth allocation effect on end-to-end flow communications.

A fully virtualized testbed was created with hosts running Hadoop. These hosts (Virtual Machines - VMs) were interconnected through a virtual network topology with spine and leaf switches. The virtual topology contains two spine switches ($ovs1$ and $ovs2$) and four leaf switches ($ovs3$, $ovs4$, $ovs5$ and $ovs6$). Each leaf switch was connected to three VMs. All virtual links were configured to 1Gbps. The spine switches links were numbered 1 to 4, from left to right. The link from $ovs1$ to $ovs3$ was named link1, the link between $ovs1$ and $ovs4$ was named link2, and so forth.

Testbed Configuration

OpenVSwitch v1.4.6 was used to create the virtual switches, and a POX controller (v0.2.0) was used to configure the switches by using OpenFlow v1.0. VirtualBox v4.1.12 was used for machine virtualization. For cloud management, Cloudera 5.1.1 was installed in the VM called CloudMaster (CM). Hadoop (v2.3.0) Yarn ResourceManager and HDFS NameNode were installed on HadoopMaster (HM). In the VMs named Slave[1-10] (S[1-10]), the Hadoop clients Yarn NodeManager and HDFS DataNode were installed.

This virtual environment was configured on a SuperMicro H8QG6 server (AMD Opteron 6136, 800MHz, 24 cores, 94GB RAM, 2x1TB HDDs SATA2) running Ubuntu 12.04, 64 bits. All VMs ran Ubuntu 12.04 64 bits, configured as following: CloudMaster with 6 virtual processors and 12GB RAM; HadoopMaster with 6 virtual processors and 9GB RAM; and all Slave with 2 virtual processors and 3GB RAM.

Experiments

To generate Hadoop traffic used in the experiments, two programs were used: TeraGen and TeraSort. They are often used to benchmark and test network bottlenecks on Hadoop clouds [8]. TeraGen generates random data to be written in HDFS DataNodes and TeraSort conducts the sorting.

In the first experiment ($exp0$) TeraGen was used to generate 1GB of data. In the first step, only one spine switch was used so, all leaf switches communicate through only $ovs1$. In the second step a NoF program (nof_{p0}) was executed to create an overlapped topology by enabling the second spine switch $ovs2$, and to modify flows generated by Hadoop connections, dividing them through both spine switches ($ovs1$ and $ovs2$). Once again TeraGen was used to generate 1GB of data. Figure 3 shows the cumulative throughput of this experiment. The cumulative throughput represents the sum of inbound and outbound traffic of all links of the switch.

In this experiment, it was also measured the execution time. In the first step, the execution time was 210 seconds. The second step, using two spine switches, TeraGen took 152 seconds. After the changes effected by NoF, it was possible to perceive that the traffic was divided between both spine switches resulting in a reduction of 27.6% in execution time.

In the next three experiments ($exp1$, $exp2$ and $exp3$) TeraGen was used to generate respectively, 100MB, 500MB and 1GB. After that, TeraSort was executed to sort the generated data. The experiments $exp4$, $exp5$ and $exp6$ show the

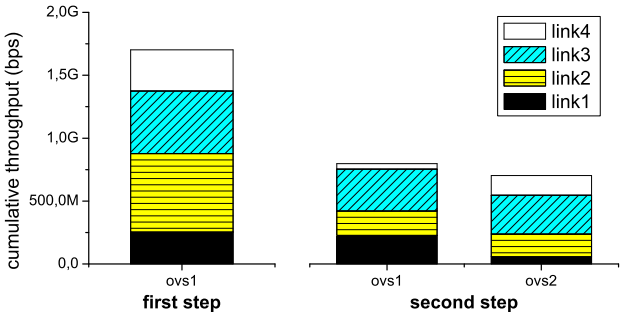

Figure 3: Cumulative throughput using overlaid topologies with one and two spine switches.

results for sorting, respectively, 100MB, 500MB and 1GB. Iperf was used to create a TCP background congestion traffic during all these experiments.

A NoF program (nof_{p1}) was created to prioritize network traffic for new Hadoop and HDFS connections. When hosts received new connections to these processes, new rules for traffic prioritization were installed. In the first step (s_1) the Hadoop jobs ran without any NoF program so, no connections were prioritized. In the second step (s_2), the program nof_{p1} was called before the execution of Hadoop jobs and, upon each new connection, QoS rules were installed to prioritize Hadoop traffic.

The experiments were repeated thirty times and the execution time was measured. The relative percentage change of each experiment$_i$ (rpc_{exp_i}) was calculated by using $\bar{s}_1$ and $\bar{s}_2$ mean values. $\bar{s}_1$ was used as the reference value ($rpc_{exp_i} = \frac{(\bar{s}_1 - \bar{s}_2) * 100}{\bar{s}_1}$). Relative percentage change and standard deviation are presented in Figure 4.

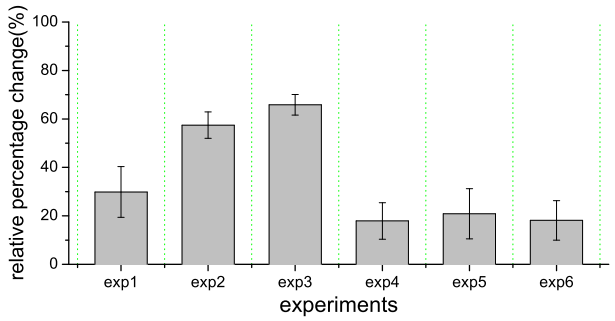

Figure 4: Relative percentage change of Hadoop execution time without prioritization compared with NoF applying bandwith guarantee.

In all experiments, there was a significant reduction in execution time after running the NoF program. The best values were obtained when Hadoop generated data (*exp*1, *exp*2 and *exp*3). The measured gain ranged from 30% to 66%. This fact occurred because there was a more intensive use of the network when HDFS writes data. In experiments where the data were sorted, there was improvement on execution time, but not too expressive, varying from 18% to 21%. The difference is explained because HDFS move Hadoop jobs closer to where the data is located [8], avoiding unnecessary transfers.

These experiments showed that NoF can be used to modify network at runtime, according to application requirements. The NoF programs created to perform this evaluation are similar to those presented in Figure 2. These abstractions offered by NoF can be easily used by specialists to meet their application requirements.

5. CONCLUSION

In this paper, we presented the Network Overlay Framework (NoF), which provides a high-level language designed to be used by application specialists. A prototype of NoF was implemented to demonstrate that the abstractions provided by NoF enable managing virtual topologies, setting the communication paths and the prioritizing the traffic. We conducted experiments in a Hadoop cloud, using NoF programs to modify network behavior, based on techniques employed by other authors to improve the Hadoop [4, 7].

As future work, NoF can be extended, and new abstractions can be developed, for example, to manage virtual machine migration and network function virtualization. An API can also be provided to allow NOF functionalities to be used directly from user applications, providing more power and flexibility in managing the network resources.

Acknowledgments: The authors would like to thanks CAPES that is providing partial funding to this research.

6. REFERENCES

[1] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory networking: An api for application control of sdns. In *Proc of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 327–338. ACM, 2013.

[2] T. Humernbrum, F. Glinka, and S. Gorlatch. A northbound api for qos management in real-time interactive applications on software-defined networks. *Journal of Communications*, 9(8), 2014.

[3] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*, 2014.

[4] I. Monga, E. Pouyoul, and C. Guok. Software defined networking for big-data science. *SuperComputing*, 2012.

[5] S. Sakr, A. Liu, D. M. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *Communications Surveys & Tutorials, IEEE*, 13(3):311–336, 2011.

[6] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster. Merlin: A language for provisioning network resources. *arXiv preprint arXiv:1407.1199*, 2014.

[7] G. Wang, T. Ng, and A. Shaikh. Programming your network at run-time for big data applications. In *Proc of the first workshop on Hot topics in software defined networks*, pages 103–108. ACM, 2012.

[8] T. White. *Hadoop: the definitive guide: the definitive guide.* " O'Reilly Media, Inc.", 2009.

[9] K.-K. Yap, T.-Y. Huang, B. Dodson, M. S. Lam, and N. McKeown. Towards software-friendly networks. In *Proc of the first ACM asia-pacific workshop on Workshop on systems*, pages 49–54. ACM, 2010.