A More Flexible SDN Architecture Supporting Distributed Applications

Wen Wang[1], Cong Liu[2(✉)], and Jun Wang[2]

[1] National University of Defense Technology, Changsha, China
[2] PLA Logistic Information Center, Beijing, China
congliu2005@163.com

Abstract. Software Defined Networking (SDN) abstracts network control logic from switches to a logically centralized controller with software implemented applications. Unfortunately, not all the applications fit the centralized control architecture, and the centralization may even degrade the performance of these applications. Moreover, even though SDN articulates a vision for programmable networks, the OpenFlow instructions with simple match-action fields restrict the flexibility of switches, and the programmability of switches has rarely been actually touched. To strike a balance between programmability and pragmatism of SDN, we propose a more flexible and powerful SDN control architecture to support distributed applications besides simple OpenFlow instructions. The distributed applications run independently in switches, and the controller is responsible for the installation and configuration of these applications. The evaluation shows the proposed architecture is able to access more local details efficiently with the centralized SDN control.

Keywords: SDN · Distributed applications

1 Introduction

Software Defined Networking (SDN) has attracted a lot of attentions in recent years with the separation of network functions, moving network control logic from switches to software applications on the controller, which results in increasing flexibility. Unfortunately, flexibility and efficiency rarely go hand in hand, as there is always a trade-off between programmability and performance. Even though switches do not need to implement protocol details, the centralized control logic may be a critical performance bottleneck. A lot of applications compete for computation and storage resources of the control plane. Moreover, applications communicate remotely with switches through control messages to manipulate flow tables, which adds to the communication overhead between the control plane and data plane. To relieve the bottleneck, a logically centralized control plane with multiple distributed controllers has been proposed, however, because of the management complexity among multiple control nodes, the scalability of SDN is still restricted.

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 165–174, 2017.
DOI: 10.1007/978-3-319-59288-6_15

While network functions in SDN have been abstracted to the centralized controller, a lot of network functions require to access distributed information from switches in real time, e.g., network monitoring, intrusion detection. Therefore, the centralized node has to collect distributed information from switches frequently. Considering the transmission overhead and limited storage space on the centralized node, fine-grained distributed information collection is prohibitive. Only coarse-grained sampling is acceptable, which usually results in relatively low accuracy.

In order to make SDN more flexible and effective, a huge number of extensions to OpenFlow have been proposed in recent years, however, the OpenFlow control messages are still limited to simple match-action instructions. The match-action instructions oversimplify the flow tables in switches, as the action field is simply forwarding, dropping or modifying packets. However, supporting a wide range of network services would require much more sophisticated functions to analyze and manipulate traffic, e.g., deep packet inspection (DPI), compression and encryption [7]. Therefore, there is no means to deploy complicated services on switches with the simple match-action instructions, so that the flexibility of SDN is still limited. Meanwhile, these simple actions underutilize the switches hardware potentiality, which equip with powerful hardware, e.g., memory, processor, and storage. Even though SDN claims a programmable network with easily deployed applications and configurable flow tables, the programmability of switches has rarely been touched actually.

To make switches more powerful to utilize the underlying hardware while maintaining the flexibility of SDN, switches should support more complicated actions besides current simple OpenFlow actions. In this paper, we extend switches in SDN to support distributed applications, so that switches are able to provide more complicated functions. The controller installs and configures these distributed applications in switches with extended application control messages. Thus, a part of previously tightly centralized control logic is released to switches. We present two types of distributed applications based on their implementations in this paper: administrator-developed applications which are executable programs developed by the administrators, and module-constructed applications which are constructed with Click [9] elements and run as lightweight VMs. The evaluation shows that distributed applications could access more local details than centralized approaches with a little overhead, and the controller is able to manage these applications efficiently.

The rest of the paper is organized as follows. Section 2 looks at the related work. Section 3 proposes the architecture supporting distributed applications. Section 4 describes implementation details, and Sect. 5 evaluates the basic performance of the architecture. Finally, Sect. 6 concludes the paper.

2 Related Work

A lot of existing researches have been aware of the insufficiency of current SDN to support complicated actions. [3,7] note that current SDN produces insufficient

abstractions with simple instructions to cover a wide range of sophisticated networking services. [5,6] indicate that the programmability and flexibility of SDN should be extended to the data plane to allow network owners to add their custom network functions. Therefore, a lot of efforts have been made to create programmable network infrastructures. [3] uses the Click modular router language to orchestrate Linux networking tools. NetOpen [8] supports configurable networking with programmable networking switch nodes. [4,13] suggest switches should support flexible mechanisms for parsing packets and matching header fields with protocol-independent packet processors. [11] proposes an extended application-aware SDN architecture with stateful actions in switches to use L4-L7 information. [12] extends SDN to control the scheduling and queueing behavior of a switch by adding a small FPGA in switches. However, these approaches either require extra modifications in switches or are little controlled by the control plane, which lose the flexibility and manageability of SDN.

3 A Switch Supporting Distributed Applications

3.1 SDN Architecture Supporting Distributed Applications

The complexity and overhead of implementing and executing all these software applications in the centralized controller motivate it to release parts of control logic to switches, especially for the applications which need to access distributed information frequently. Therefore, these software applications should be distributed into multiple locations for advanced performance, while being managed by a logically centralized controller.

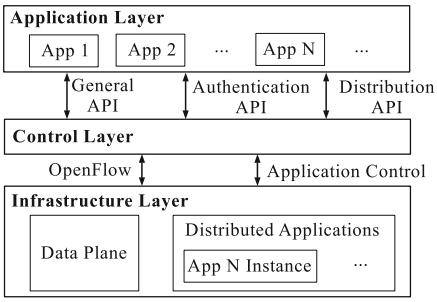

Fig. 1. System architecture

To support these distributed applications, the infrastructure layer not only acts as a data plane, but also runs instances of distributed applications in switches. These distributed instances could be programs transferred and installed by the controller, and run in switches with local fine-grained information. A distributed application instance is able to execute independently and usually does not need to communicate with instances in other switches, but may require

to contact with the controller when necessary. Hence, in spite of OpenFlow to manipulate flow tables in switches, extended application control messages are used between the control layer and the infrastructure layer to install, configure and communicate with distributed application instances in switches as Fig. 1 shows. These messages enable distribution applications to be dynamically programmed in network infrastructures so as to enhance the scalability and flexibility of SDN. Therefore, administrators are able to develop their own distributed applications and then dynamically install them in switches.

Even though there is no standard northbound API currently, we consider that SDN deploys centralized applications on the controller with general APIs, and we only focus on the distributed applications in this paper. The distributed applications need to define the distribution features such as concerned flows, executable programs, initial parameters with a distribution API, which is unnecessary for centralized applications. With distributed instances running in switches, more switch details are exposed to these distributed applications. However, as applications are able to manipulate flow tables in switches to control network behaviors, poorly implemented, misconfigured or malicious applications may modify flow tables deliberately. Therefore, the controller has to ensure the legitimacy of distributed applications to prevent abnormal activities. Unfortunately, the absence of the northbound API fails to limit the access authorities granted to applications. Thus, an authentication API is added to verify the access permissions of distributed applications between the application layer and control layer. To ensure the legitimacy of distributed applications, the controller has a white list record of legal distributed applications. Distributed application control requests should be issued by authenticated applications, otherwise, requests to manipulate distributed applications in switches will be rejected by the controller.

3.2 Distributed Applications in SDN Switches

A lot of SDN applications running in the controller require to communicate with switches frequently for fine-grained information collection or data plane control. These applications obviously need to deal with distributed information in distributed architectures, e.g., network monitoring, intrusion detection, while the current SDN manages all the applications as centralized. To distinguish distributed applications from centralized ones, we define the criteria of a distributed application which is appropriate to be deployed distributedly in switches:

- Access local fine-grained information such as traffic statistics or packet payload frequently, and do not need to wait for other remote data or control messages.
- There are few control message exchanges with the controller. The controller just needs to set up the distributed application in switches at the beginning, and then a switch is able to execute the distributed instance independently.
- Require configurations or updates occasionally, so that the controller manages a distributed instance with extended application control messages instead of proprietary application implementations on switches.

– Execute complicated functions instead of simple OpenFlow actions, and the complicated functions could be triggered by sophisticated conditions other than the simple match field of the flow table.

Due to the remote installation and configuration, distributed applications should be carefully designed to ensure the correctness and effectiveness during execution. Considering the construction and implementation of applications, in this paper, we present two types of distributed applications: administrator-developed applications and module-constructed applications.

Administrator-Developed Applications. These applications are executable programs developed by the administrators and then deployed in switches with control messages. For the application management, the controller transfers the executable programs to the switch with control messages and installs a corresponding entry for each application in the application table. The executable programs are recorded in the disk of the switch, so that distributed applications will not be lost when switches reboot. Application table records the application entries and related programs. When packets matching the application entry arrive, the corresponding programs are executed in the execution engine. The execution engine uses memory and processor of the host switch, so that the hardware is highly utilized with various distributed applications. When an application is being executed, it may need to access information in the flow table or capture packet payload, e.g., DPI. Therefore, we also design interfaces between the execution engine and flow tables.

Module-Constructed Applications. These applications are running in ClickOS [10] VMs assembled with modules in switches. ClickOS is a Xen-based tiny virtual machine that runs Click [9], and it can be quickly instantiated in 30 *ms* with a compressed 5 MB image. As Click equips with over 300 stock elements, which make it possible to construct applications with minimal efforts. Therefore, module-constructed applications could be assembled with these elements in virtual machines to be ClickOS VMs. Moreover, we can easily extend this framework and develop new elements with the administrators-developed to support more applications. To set up a module-constructed application in ClickOS VM, the controller dispatches a Click configuration to related switches, which is essentially a text file specifying elements. Once receiving the configuration file, the switch instantiates a VM for the application based on the defined configuration. As applications are isolated into multiple fast booted VMs, they do not interfere with each other during processing.

As distributed applications run locally in switches, these applications are able to execute in real time with detailed local data, which is impossible for the centralized controller to perform such fine-grained controls. As the application table is separated from the flow table, it does not affect the flow table lookup efficiency. Moreover, the distributed applications are restricted with isolated hardware resource (e.g., CPU, memory) for the both types of applications, so that

the extended lightweight functions do not affect the basic packet processing of the data plane, which means the extended programmability does not decrease the packet processing efficiency.

4 Implementation

In this section, we design and implement control messages and execution engines for the two types of distributed applications with Open vSwitch [1].

4.1 Distributed Application Control

We implement two kinds of control messages for distributed applications to communicate between the controller and switches APP_MOD and APP_REP. APP_MOD is used to set up or update distributed application in switches. For the administrator-developed applications, despite the distributed programs, the control message also transfers the initial parameters for the programs together. For the module-constructed application, the controller sends the Click configuration to a switch, so that the configuration is used to instantiate a ClickOS VM. When an application instance becomes expired or loses effectiveness, the controller could remove it by deleting the corresponding application entry and removing related programs or shutting down VMs in the switch using APP_MOD messages. During the execution of an application, if it would like to communicate with the controller, it sends APP_REP messages to the controller.

4.2 Execution Engine

Administrator-Developed Application Execution. As the administrator-developed applications are executable programs running in execution engine, we implement an execution engine supporting programs developed in C, JAVA and MATLAB. The programmed functions could be triggered by the arrival of packets or run periodically every a short interval, which is decided by the programs developed by the administrators. Thus, these applications can capture finer-grained details than centralized schemes. The local information on switches utilized by administrator-developed applications could be divided into three categories: traffic statistics (e.g., packet count, flow duration), packet sampling which capture and analyze packet header or payload, and other local information of switches (e.g., CPU and memory utilization).

Module-Constructed Application Execution. The module-constructed applications are constructed with various Click elements, e.g., IPRateMonitor, TCPCollector, Classifier. The variety of Click elements allows applications in VMs to perform diverse complicated functions in addition to simple OpenFlow actions in the flow table. The module-constructed applications are isolated into VMs with restricted memory and CPU resources, and ClickOS accesses packets with a direct pipe between NIC and VMs [10]. Therefore, applications in VMs do not affect the basic efficiency of data plane packet processing, while OpenFlow handles regular requests to manipulate the flow table as usual.

5 Evaluation

The distributed applications not only reduce communication overheads between the controller and switches, but also relieve administrators from heavy labour configuring work by deploying and controlling these distributed applications with the centralized controller. To show the efficiency of the proposed architecture, we evaluate the execution and management performance of the two types of distributed applications. As the application performance greatly depends on the design and implementation of each application, we mainly focus on information collection performance, throughput and management overhead of these distributed applications.

5.1 Distributed Information Collection Performance

As distributed applications usually utilize local information for network monitoring or anomaly analysis, we evaluate the collection efficiency of the three kinds of local information in distributed applications and compare them with centralized approaches.

(a) Traffic Statistics: As OpenFlow provides control messages to poll traffic statistics from switches, the centralized controller usually uses periodical polling which collects traffic statistics every several seconds. With the remote polling, the fetching delay of port statistics using OFPMP_PORT_STATS messages is almost 400 μs, while the distributed application is able to access the statistics locally within 13 μs in Table 1. Moreover, the periodical interval is difficult to decide for different statistics granularity, and the communication overhead also depends on the polling interval and grows linearly with the polling frequency. On one hand, the smaller the collecting interval is, the larger the overhead is. On the other hand, if the collecting interval is quite large, it may miss a lot of short abnormal details because of the coarse monitoring granularity. Hence, it is hard to strike a balance between the statistic overhead and accuracy in centralized approaches, which is not a problem in distributed schemes.

(b) Packet Payloads: Due to the large network traffic volume, network monitoring applications which analyze packet payload (e.g., DPI) usually capture packets with sampling. Compared with the centralized packet capturing approaches, the distributed applications in switches do not need to transfer packets to a centralized node which saves a lot of transmission delay for real-time analysis. We compare our distributed packet payload capturing with sFlow [2], and the result shows that the centralized approach takes over 7×10^5 μs which is almost 9000 times larger than 86 μs locally in switches to fetch packet payloads. Furthermore, the communication overhead and limited memory space in a centralized node also restrict the performance of centralized packet payload capturing and inspection. Similar to the traffic statistic, the collected packet payload size also grows proportional to the sampling rate, which brings a great overhead for centralized fine-grained packet capturing.

(c) CPU/Memory Information: With distributed instances running in switches, these instances are able to access more local information with assigned permissions, such as CPU and memory utilization. The latency to access CPU and memory statistics with system files /proc/stat and /proc/meminfo in Ubuntu is about 14 μs while it is unavailable for centralized approaches.

Table 1. Information collection latency (μs)

	Distributed	Centralized
Traffic statistics	13.14 ± 1.96	405.11 ± 93.24
Packet payloads	86.41 ± 12.17	$7.535 \times 10^5 \pm 1.204 \times 10^5$
CPU/Mem info	14.12 ± 0.91	NA

In spite of the shorter latency of information collection in distributed applications, they also transfer less control messages between the controller and switches than centralized schemes. The controller just needs to transfer programs or configurations at the setup of distributed applications, and distributed instances could then execute independently.

5.2 Throughput of Distributed Applications

As the performance of distributed application strongly depends on the design and implementation of programs, we only evaluate the throughput of these applications by injecting related packets. The switch is equipped with a 1Gb/s connection. We use an administrator-developed application to get statistics from the flow table. The throughput is almost closed to line rate in Fig. 2, as the statistics collection between the execution engine and the flow table does not affect the basic packet processing. For the module-constructed applications, we evaluate an application constructed with element *Counter*, and the throughput achieves at least 80% injection rate. The optimized I/O pipe of ClickOS helps to improve the throughput [10], which means simple ClickOS configurations add little overhead.

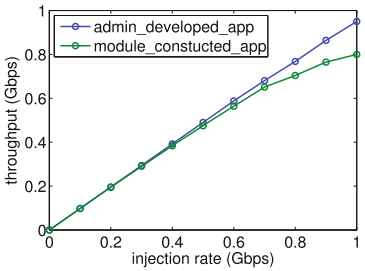

Fig. 2. Throughput of distributed applications

5.3 Distributed Application Management Overhead

To show the efficiency of distributed application management, we evaluate the application transmission and configuration latency with the increasing of the program/configuration file size and the network size respectively. As switches are connected directly to the controller, the application management is independent from the network topology. We test a 2-D mesh network with Mininet, and each switch in the network connects to the controller with a 1 Gb/s link.

To execute multiple applications in switches efficiently, the administrator-developed applications are usually small-sized lightweight programs. Meanwhile, as the configuration file of module-constructed applications only needs to define the element names and rules with integrated elements in ClickOS, the size of configuration file is also quite small at the level of kilobytes. In Fig. 3, when the size of transferred program/configuration file grows, the distribution latency also increases, and it takes over 20 ms to send a 10 Mb program/configuration file to a switch. Nevertheless, it is still acceptable for the overall lifetime of a distributed application, as the controller only transfers programs or configuration files at the beginning. In the proposed SDN architecture supporting distributed applications, when a switch sets up a distributed instance, it inserts a corresponding entry in the application table and records the programs/configuration file on the disk. The application table latency is quite short as the result shows. The administrator-developed application executes the corresponding programs, while a module-constructed application boots a ClickOS VM. We notice that the ClickOS VM booting takes about 30 ms and does not increase a lot when the configuration file size grows. The overall setting up time of a module-constructed application is less than 100 ms for a 10 Mb configuration file. Thus, the setting up latency is quite acceptable to relieve the centralized controller from frequent information fetching.

Meanwhile, when the network size scales, the number of switches running distributed applications is expected to increase to relieve control logic overhead. The overhead of distributing and managing distributed instances in switches also grows as Fig. 4 shows. It takes about 250 ms to distribute a 100 Kb program/configuration file to 100 switches at once. The latency is still much shorter than the collecting and sampling interval in centralized approaches, which usually perform at the level of several seconds. Therefore, the distribution and management overheads of distributed applications are reasonably acceptable.

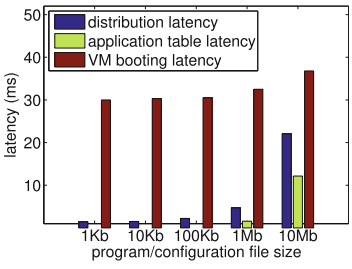

Fig. 3. Scalability with app size

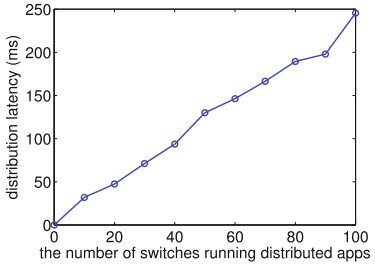

Fig. 4. Scalability with network size

6 Conclusion

Considering the dumbness of switches and the simpleness of OpenFlow actions in current SDN, we propose an extended OpenFlow-enabled switch architecture to support distributed applications in addition to simple match-action OpenFlow instructions. Therefore, a lot of previously centralized applications could be deployed as distributed instances in switches, e.g., network monitoring, intrusion detection, etc. The distributed applications do not mean distributed control logic, as the controller is still controlling these distributed instances with application control messages. The evaluation shows that distributed applications could access more local information efficiently than centralized schemes, while the controller manages these distributed applications with low overheads.

References

1. Open vswitch. http://openvswitch.org/
2. sflow. http://www.sflow.org/
3. Bhatia, S., Bavier, A., Peterson, L.: Wanted: systems abstractions for SDN. In: HotOS (2013)
4. Bosshart, P., Daly, D., Gibb, G., et al.: P4: Programming protocol-independent packet processors. In: SIGCOMM (2014)
5. Farhad, H., Lee, H., Nakao, A.: Data plane programmability in SDN. In: ICNP (2014)
6. Farhadi, H., Du, P., Nakao, A.: User-defined actions for SDn. In: CFI (2014)
7. Feamster, N., Rexford, J., Zegura, E.: The road to SDN: an intellectual history of programmable networks. In: SIGCOMM (2014)
8. Kim, N., Yoo, J.-Y., Kim, N.L., Kim, J.: A programmable networking switch node with in-network processing support. In: ICC (2012)
9. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The click modular router. ACM Trans. Comput. Syst. **18**, 263–297 (2000)
10. Martins, J., Ahmed, M., Raiciu, C., et al.: Clickos and the art of network function virtualization. In: NSDI (2014)
11. Mekky, H., Hao, F., Mukherjee, S., Zhang, Z.-L., Lakshman, T.: Application-aware data plane processing in SDN. In: HotSDN (2014)
12. Sivaraman, A., Winstein, K., Subramanian, S., Balakrishnan, H.: No silver bullet: extending SDN to the data plane. In: HotNets (2013)
13. Song, H.: Protocol-oblivious forwarding: unleash the power of sdn through a future-proof forwarding plane. In: HotSDN (2013)