Lossless Reconfiguration Protocol for Multi-Domain Data Plane in Software-Defined Networks

Boyang ZHOU†, Haifeng ZHOU†, Wen GAO†, Xiaoyan HONG*, Bin WANG† and Chunming WU†♯
College of Computer Science, Zhejiang University, Hangzhou 310027, China†
Department of Computer Science, University of Alabama, AL 35487, USA*
{zby,gavingao,bin_wang,wuchunming}@zju.edu.cn, hf.zhou.ccnu@gmail.com, hxy@cs.ua.edu

I. Introduction

Software-Defined Networking (SDN) opens the configuring interfaces for the data plane to improve its flexibility[1]. In the data plane, the switches are partitioned into multiple domains. Each domain is dynamically configured by its controller. The domains are physically connected via the border switches. In the control plane, each controller supports multiple services. Each service runs the same control program at multiple controllers in a distributed manner.

When a service changes its forwarding rules of the border switches in the data plane, multiple controllers will reconfigure their corresponding border switches. The reconfiguration messages from different controllers usually reach their border switches in a random order due to the different delays encountered. This asynchronous communication leads to inconsistent states at the border switches, the service is then in the transient state where packet losses and flow interruptions can happen, disrupting its availability. Examples include hiccups of Voice-over-IP (VoIP) and server connection losses, etc.

The current work in the area of Internet research solve the problem by either ordering the steps of reconfiguration[2] or migrating the resources of routers to reduce packet loss rate[3]. But there are still negative effects: the in-flight packets can still be discarded in executing each ordered step; and migrating the resources is a costly operation for the data plane.

No existing work has exploited the unique advantages of SDN to solve the transient-state-induced availability problem of the network services running using SDN. For example, the standard platform of the control plane offers opportunities to manage the states of all the switches in the current domain already. The state-of-the-art in SDN research, especially those on the current realizations of the SDN controllers have not dealt with the transient state problem for interaction between the controller plane and the data plane where reconfigurations for the border switches are concerned mostly.

In this poster, we propose a lossless reconfiguration protocol implemented in the control plane to solve the transient state problem. Our protocol is light-weight and ensures that there is no in-flight flow belonging to the service during the reconfiguration process. Using the protocol, services or operators can reconfigure the states on the border switches into new states with minimum in-flight packet loss rate. They will not need to separately devise mechanisms to take care of the problem by themselves. In addition, we introduce the supervision layer within the controller which implements the protocol and also holds the various states in reconfiguration.

♯ corresponding author, E-mail: wuchunming@zju.edu.cn

This work is supported by the National Basic Research Program of China (973 Program) (2012CB315903), the Key Science and Technology Innovation Team Project of Zhejiang Province (2011R50010-05) and the National Natural Science Foundation of China (61379118 and 61103200). This work is sponsored by the Research Fund of ZTE Corporation, and also supported in part by the BBN/NSF Project 1783.

II. Protocol and Implementation

A. Reconfiguration Protocol

The reconfiguration protocol minimizes the loss rate of the in-flight packets when applying new forwarding rules to the border switches. Such problem is solved by first storing the in-flight flows of all the border and non-border switches before applying the forwarding rules, and then restoring these flows after applying. We term two types of states: (i) the *flow states* are the states of in-flight packets for a flow, i.e., flow traffic, and (ii) the *behavioral states* are the states that decide the forwarding behavior of border switches, e.g., forwarding rules.

The protocol is initialized from a controller to make a group of controllers updating their forwarding rules of their border switches in their domains. Each controller c_i has three major phases that are strictly separated with each other:

Phase 1: Storing the flow states. In Phase 1, c_i first suspends the process of the current service am_i; then sends the extended OF command to suspend the forwarding of the flows at the switch; finally it stores the flow states of the service of all the border and non-border switches in its own domain D_i. To ensure that all the in-flight packets are all captured, the switch waits for the time period $-ln(1-p)\tau$, where p and τ are confidence possibility and average link delay, since most flows are in the Poisson distribution, so that the last packet sent with possibility p before suspension arrives at the forwarding queue. It then stores the flow states.

Phase 2: Updating the behavioral states. For c_i, upon receiving a set of updates of the behavioral states from D_i, it first merges these behavioral states into R as the new states. It then updates the new states into the forwarding resource of the corresponding border switches in its domain D_i.

Phase 3: Restoring the flow states. c_i first sends the extended OF command to all the border and non-border switches in the D_i to resume the forwarding of flows of the service and restores the flow states of the service. It then resumes am_i.

When using the minimum spanning tree to disseminate collaboration states among controllers, the message and time costs of overall execution of the protocol are $O(D+d)$ and $O(E+m\times d+m^2)$, where m, l, E, D and d are the number of intra-domain switches, the average length of queues in the switches, the number of physical links in the domain, the diameter of the domain-level topology of networks by counting the links as well as the max domain diameter for all domains.

B. Supervision Layer

The reconfiguration protocol is implemented as a primitive in the new layer named as supervision layer. The layer runs

978-1-4673-1017-8/14/$31.00 ©2014 IEEE

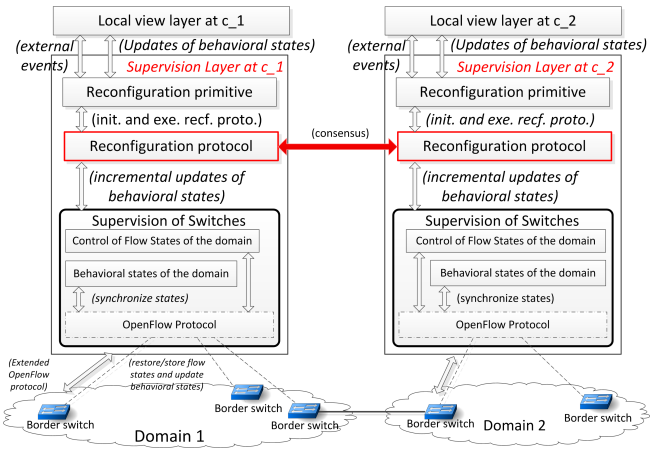

Fig. 1. Architecture of Supervision Layer

between the two functions originally implemented by the NOX controller[4] (see Fig.1): (i) the OpenFlow (OF) protocol realizing the OF message stack, and (ii) the local views contains all device OF states of border switches, e.g., port throughput as well as adjacency states from link layer discovery protocol. The layer supervises the operations of writing or reading the behavioral states and the flow states of the border switches at all the intra-domain switches, elaborated as follows.

1) Update aggregation of behavioral states: Update aggregation ensures both efficiency and correctness of applying behavioral states. For the efficiency, the supervision layer aggregates the recent updates and compresses into a single reconfiguration operation. To this end, the controllers are organized using a standard synchronization mechanism with a leader, so that all the controllers are in the same aggregation round with the same set of aggregated updates when the reconfiguration is triggered by the leader. For correctness, when the new behavioral states are received from a service, these states are detected for their collisions with other services'according to their matching priority.

2) Storing and restoring the flow states: All the flow states of all the switches are maintained at the switch side in a distributed way to achieve efficiency. We extends two OF architectures: (i) The switch architecture is extended with a new extra memory for temporarily saving the flow states based on DRAM. Besides, the current industrial NEC PF5240 switch proves feasibility to store flow header states in its buffer[5], thus it is also feasible to store these packets in the buffer of the switch considering these packets being widely distributed in the links of its original route and the time of reconfiguration being transient. And (ii) The OF protocol v1.0[1] is extended to convey commands of storing and restoring the flow states. The extension is used to negotiate between the controller and a switch to control reading and writing these memories, by carrying the commands within "packet_out" and "packet_in" OF messages.

III. Evaluation

The protocol is evaluated for a ICN prototype realized as a SDN service. The prototype is based on CCNx[6], where the forwarding information bases (FIBs) of ICN is implemented as the flow table at the switch side, and the rest of ICN functions are at the controller side. When the ICN's content routing reconfigures on multiple FIBs, the in-flight Interest packets can be discarded when those packets are forwarded in loop due to asynchronous updates for the FIBs, causing the solicited

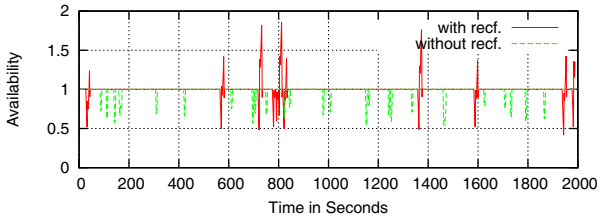

Fig. 2. Availability of Service Flow for ICN Service

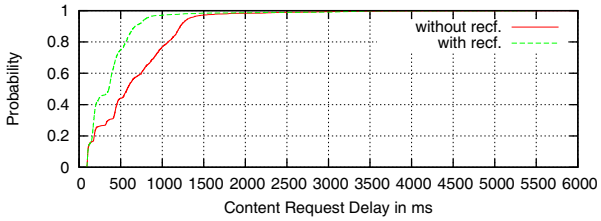

Fig. 3. CDF of Data Delay for ICN Service

data packets lost in downstream[6]. The effectiveness of the protocol is evaluated on the PlanetLab testbed[7] with 9 nodes of high bandwidth, where three ICN consumers send Interest packets at 1000 per second in Poisson to a node along rest of nodes. p is set to 0.95 (see Subsection II.A). Fig. 2 shows the availability ratio (equaling to the number of Interest packets divides the number of data packets) is averaged to 1 for our prototype, where the spur points have values larger or less than 1 (the solid red line), while the raw prototype is less than 1 that data packets are lost in updating. Averaging on simulation time of 3000s, Fig. 3 shows the performance comparison in terms of the cumulative density functions of delay for the two versions. Taken 95% of delay distribution as an example, the mean values are 807ms and 1317ms for our prototype and raw implementation. The performance gain is clearly demonstrated in comparison to the latter one.

IV. Conclusion

We proposed a reconfiguration protocol and an associated supervision layer in the controller to preserve in-flight flows in reconfiguring, improving service availability. We demonstrate the benefits of the supervision through the experiments using a customized ICN prototype. The results from the PlanetLab testbed show that ICN can maintain the continuity of service flows when updating its behavioral states of ICN switches.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan *et al.*, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] S. Vissicchio, L. Vanbever, C. Pelsser *et al.*, "Improving network agility with seamless bgp reconfigurations," *IEEE/ACM Transactions on Networking*, vol. 21, no. 3, pp. 990–1002, 2013.

[3] L. Vanbever, J. Reich, T. Benson *et al.*, "Hotswap: Correct and efficient controller upgrades for software-defined networks," in *ACM SIGCOMM Workshop on HotSDN*, 2013.

[4] N. Gude, T. Koponen, J. Pettit *et al.*, "Nox: towards an operating system for networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[5] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton *et al.*, "Networking named content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, 2009.

[7] B. Chun, D. Culler, T. Roscoe *et al.*, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.