2015 IEEE 35th International Conference on Distributed Computing Systems

LazyCtrl: Scalable Network Control for Cloud Data Centers

Kai Zheng*, Lin Wang†, Baohua Yang*, Yi Sun†, Yue Zhang*, Steve Uhlig‡
*IBM Research
†Institute of Computing Technology, Chinese Academy of Sciences
‡Queen Mary University of London

***Abstract*—The advent of software defined networking enables flexible, reliable and feature-rich control planes for data center networks. However, the tight coupling of centralized control and complete visibility leads to a wide range of issues among which scalability has risen to prominence. We observe that data center traffic is usually highly skewed and thus edge switches can be grouped according to traffic locality. As a result, the workload of the central controller could be highly reduced if we carry out distributed control inside those groups.**

Based on the above observation, we present LazyCtrl, a novel hybrid control plane design for data center networks. LazyCtrl aims at bringing laziness to the central controller by dynamically devolving most of the control tasks to independent switch groups to process frequent intra-group events using distributed control mechanisms, while handling rare inter-group or other specified events by the controller. We implement LazyCtrl and build a prototype based on Open vSwich and Floodlight. Trace-driven experiments on our prototype show that an effective switch grouping is easy to maintain in multi-tenant clouds and the central controller can be significantly shielded by staying lazy, with its workload reduced by up to 82%.

I. Introduction

The routing and forwarding protocols designed for current data centers are restricted to very specific deployment settings, leading to inflexible configuration and management. However, this situation has been revolutionized by Software Defined Networking (SDN), where the network control plane, separated from the data plane, is implemented with a logically centralized controller. Taking advantage of centralization, flow-based polices can be conveniently applied to achieve fine-grained control over the data center network.

While becoming more and more prevalent [1], [2] , flow-based centralized control has given raise to some new challenges, one of which is the scalability issue brought by the excessive coupling of central control and complete visibility. It has been demonstrated that full control and visibility over all flows are not always necessary and devolving some control authority to the data plane by proactively suppressing frequent events can result in better scalability in SDNs [3]. However, the right granularity for flows to be handled by the controller remains blurred. In this work, we advocate a new approach for control devolvement for software defined data center networks (DCNs) based on traffic locality. The idea steams from the observation that traffic distribution in data centers (especially those with multi-tenancy support) could be highly skewed, i.e., frequent communications are more likely to take place inside certain small groups of hosts [4]. As a result, it is possible to shield the central controller from many frequent events inside these groups with distributed control mechanisms.

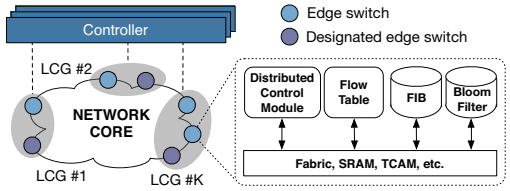


Fig. 1. Architecture design of LazyCtrl, where the network control plane consists of a logically centralized controller and distributed control modules in multiple LCGs.

We propose LazyCtrl, a hybrid network control plane design for cloud data centers, which seeks to bring laziness to the central controller. In our design, edge switches forms logical groups dynamically according to their communication affinity. A central controller devolves the coarse-grained control for frequent intra-group events to each switch group while handling infrequent inter-group and other specified (fine-grained) control tasks by itself. Each switch group autonomously carries out distributed control within the group, keeping the intra-group packets in the data plane. The controller groups the switches in such a way that the size of each group is as large as possible to exhaust switches' memory (such as TCAMs) capacity while inter-group traffic is minimized to support the laziness of the controller.

We completed a full implementation of LazyCtrl and built a prototype to validate the performance of the design. Experiences on our prototype with both real and synthetic traffic traces show that an effective switch grouping is easy to maintain in multi-tenant clouds and the hybrid control design can highly reduce the workload of the controller and provides lower latency in packet forwarding. As expected, the laziness we introduced to the controller decouples centralized control and complete visibility and consequently, it can scale the system much better compared with totally centralized designs. This abstract only provides the basic idea and more detailed information could be found in the full version of the paper [6].

II. The Design

In conventional flow-based centralized control environment such as those based on OpenFlow [5], the controller maintains the network-wide state (i.e. the host-to-switch mapping) and handles all the flows between every pair of switches that exchange data, bringing extremely high burden to the controller. LazyCtrl mitigates this problem by clustering the switches into multiple switch groups according to their communication affinity and devolving intra-group control to these switch groups (termed Local Control Groups, LCGs).

1063-6927/15 $31.00 © 2015 IEEE
DOI 10.1109/ICDCS.2015.110

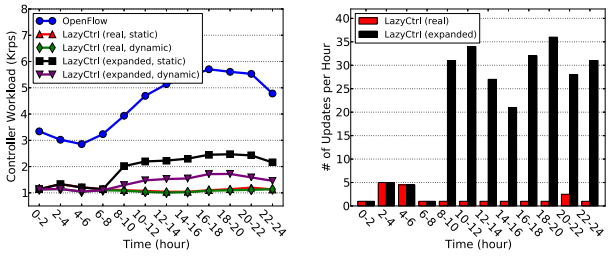
The architecture design of LazyCtrl is depicted in Fig. 1. Our design splits the core from the edge. The network core can be any simple and scalable network (e.g. an IP unicast network) which serves as the underlay providing connectivity for the edge switches. The network edge is in charge of network intelligence, i.e., resolving host-to-switch mappings.

We introduce a hybrid control model which involves a central controller and a set of LCGs. The central controller has holistic visibility over the entire data center network and is responsible for i) maintaining a Central Location Information Base (C-LIB) which preserves host location information, ii) adapting the grouping of the edge switches, and iii) managing the flow tables on the edge switches to handle inter-group traffic and any specific traffic that needs flexible centralized control. The goal of the central controller is to stay lazy by devolving as many control tasks as possible to the LCGs. The central controller can be a standalone physical server or a logical controller comprised of a cluster of servers with strong reliability and coherency of network state.

An LCG is a group of edge switches whose clients are observed to have frequent mutual communication. These switches are grouped together by the controller and share the network state with each other continuously . Each LCG employs a distributed control mechanism to take over the control workload of intra-group traffic from the controller. The distributed control mechanism inside each group is carried out by equipping each edge switch with some local forwarding tables that are realized by Bloom filters. These local forwarding tables keep track of network states such as host-to-switch mappings inside the corresponding group. For each LCG, a designated switch (with some backups) is selected randomly by the controller, which is responsible for aggregating group-wide network states from the edge switches in this group and reporting them to the controller in an asynchronous manner. Each edge switch also maintains a forwarding information base for identifying local hosts.

The design of LazyCtrl is based on the concept of grouping switches to form multiple LCGs. Thus the quality and efficiency of the grouping is essential to the whole design. Given a limit for the group size (determined according to empirical or historical data), a good grouping scheme is defined as one in which the inter-group traffic is small (in order to facilitate the laziness of the controller) and the computational complexity of the grouping algorithm is sufficiently low such that it can fast adapt to traffic dynamics. Our grouping algorithm aims at satisfying the above principles. We base our design on solving the classical graph partition problem using a modified Multi-Level k-way Partition (MLkP) algorithm where we introduce extra constraints on partition size. Besides, we also propose a greedy refinement process based on group rejoining and splitting to incrementally update the switch grouping when facing traffic variations. This process eliminates the necessity of running the grouping algorithm entirely every time when traffic changes and thus can largely reduce the time complexity for switch grouping updates in traffic dynamics.

Failover We propose a self-detection mechanism to handle failures in the control plane based on a group-wide failure detection wheel with the controller at the center and the switches at the edge. To detect failures, keep-alive messages will be initiated from upstream switches to downstream switches and from the controller to each switch. We provide failover mechanisms for both link and switch failures by having alternatives for paths and switches.



(a) (b)

Fig. 2. Evaluation results: (a) controller workload, (b) update frequency.

III. Evaluation

We implement LazyCtrl by extending the OpenFlow protocol and developing edge switches and the controller based on Open vSwitch [7] and Floodlight [8]. We build a prototype system with 6 Pronto 3290 switches and 24 IBM x3550 8-core servers following the aforementioned design details. We replay the traffic traces collected from a multi-tenant data center in Europe on our prototype and evaluate the performance of the proposed solution.

We validate the effectiveness of LazyCtrl by measuring the controller workload under traffic dynamics and the experimental results are illustrated in Fig. 2(a). We make the following observations: i) LazyCtrl can help achieve a significant level of workload reduction (61% - 82%) for the controller. ii) The controller workload in LazyCtrl is relative stable due to the fact that most of the traffic growth is irrelevant to the controller. iii) The update function we designed for the switch grouping algorithm can successfully handle traffic dynamics. We also notice from Fig. 2(b) that the update frequency stays at a very low level (around 10 updates per hour), indicating that maintaining a relatively effective grouping is feasible in practice.

References

[1] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: saving energy in data center networks," in *NSDI*, 2010, pp. 249–264.

[2] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: fine grained traffic engineering for data centers," in *Co-NEXT*, 2011, pp. 8–20.

[3] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: scaling flow management for high-performance networks," in *SIGCOMM*, 2011, pp. 254–265.

[4] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *IMC*, 2010, pp. 267–280.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. M. Parulkar, L. L. Peterson, J. Rexford, S. Shenker, and J. S. Turner, "Openflow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] K. Zheng, L. Wang, B. Yang, Y. Sun, Y. Zhang, and S. Uhlig, "LazyCtrl: scalable network control for cloud data centers," arXiv, `http://arxiv.org/abs/1504.02609`.

[7] Open vSwitch. `http://openvswitch.org/`.

[8] Floodlight. `http://floodlight.openflowhub.org/`.