Poster: Application-Aware Load Migration Protocols for Network Controllers

Sepehr Abbasi-Zadeh, MohmmadAmin Beiruti, Yashar Ganjali
Department of Computer Science, University of Toronto, Canada
{sepehr, beiruti, yganjali}@cs.toronto.edu

Zhenhua Hu
Huawei Research, Canada
zane.hu@huawei.com

***Abstract*—Load migration protocols have been used for load balancing in network controllers. In this poster, we argue that other network applications (e.g., power saving, network security, failure recovery, etc.) have properties that might require different load migration protocols. We introduce four new load migration protocols and show how they might match different application requirements better. We present preliminary experimental results for one of these protocols that show more than $20\% - 30\%$ speedup in the total load migration time.**

I. Introduction

Decoupling control and data planes has led to agile and flexible network control and management solutions in Software-Defined Networking (SDN). Various distributed control platforms (e.g., [1]) are introduced in the last few years to address well-known reliability and scalability issues associated with physically centralized network controllers.

Load Migration in SDN Controllers. A direct consequence of moving towards distributed controllers is the need for distributing network control load amongst different controller instances. This is usually achieved by migrating the controller load as the network traffic and conditions change.

Load balancing in distributed systems is a classic problem, and has also been studied in the context of SDN controllers (e.g., [2], [3]). The solutions in this domain, mainly focus on efficient protocols for moving the load away from highly loaded controller instances towards lightly loaded instances. These solutions exchange the master and slave controller instances associated with a given switch, which results in shifting the load away from the initial master controller and towards the final master (or the initial slave controller associated with the switch). More specifically, let us consider a switch S with a master controller instance C_1, and a slave controller instance C_2. If the load of C_1 is higher than C_2, we can exchange the role of master and slave controller instances, turning C_1 into a slave, and C_2 to the master. Since slave mode controller instances do not receive and respond to all messages from S, this will result in a load reduction in C_1 (and an increased load in C_2).

Other Applications for Controller Load Migration. Load balancing is an important network control application, but it is not the only application that requires shifting controller load. Many control applications—such as power saving, network security, failure recovery, network monitoring, and controller updates—require load migration as a building block. Each of these applications, however, has a different set of requirements when it comes to switch migration compared to load balancing. For example, power saving requires shifting traffic away from lightly loaded controller instances, so that we can turn off these instances and save power. This is the opposite of what load balancing solutions try to do. Or, for faster failure recovery, network operators may decide to bring up several controller instances in equal mode, ready to serve when a given master fails.

New Protocols. In this poster, we argue that exchanging master and slave controller instances of a given switch are not sufficient, and we need other protocols that match other network control applications. In particular, we propose four new protocols (in addition to the existing master and slave exchange protocol) for switch migration that we believe cover a range of network applications.

As a proof of concept, we simulate a migration scenario under various load settings for one of these protocols and show how this application-based view of migration protocols can outperform existing solutions.

II. New Protocols

In this section we introduce four different migration protocols as well as some of their potential use cases. We note that all these protocols satisfy the five required properties of *liveness*, *serializability*, *safety*, *consistency*, and *failure resiliency* as described in [3]. Due to space limitations, we only present the full specifications of the first two protocols and leave the others for the final poster.

- *Protocol 1*: initial master controller demotes itself to equal mode and another controller in slave mode changes its role to master (Figure 1). This scenario does not change the load of the master but leads to a new equal mode controller instance. It can be used for resource optimization, and can be used to enhance the resiliency of the control plane to failures.
- *Protocol 2*: symmetric role exchange between an equal mode controller and another controller in slave mode (Figure 2). In other words, initial equal controller changes to slave mode, and another slave controller turns to equal mode. Since the master controller instance is not touched, we can use this protocol to shift traffic away

978-1-7281-6992-7/20/$31.00 ©2020 IEEE

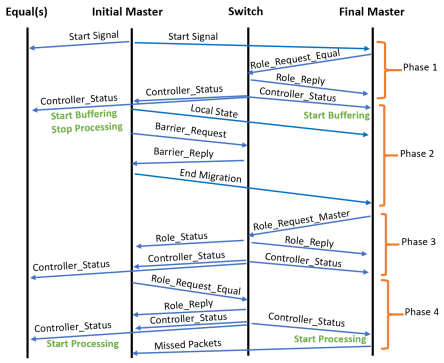

Fig. 1. Protocol 1: a master controller demotes to an equal and another slave controller becomes the master.

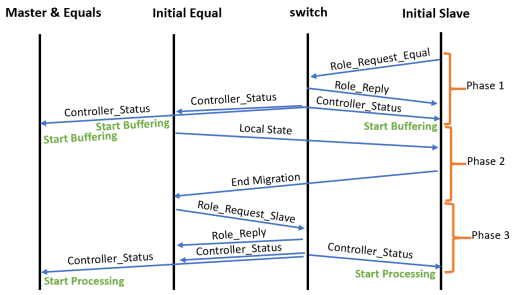

Fig. 2. Protocol 2: a slave controller takes the responsibilities from an equal controller, and this equal controller turns to its slave mode.

from one controller instance to another without impacting the normal operation of the switch.

- *Protocol 3*: initial master controller changes its role to slave and another equal controller takes the responsibility of master role. This is a perfect building block for quick shifting of the load away from the master. Since we turn an equal mode controller into master, which is relatively fast, the network can resume its normal operation faster.
- *Protocol 4*: initial master controller downgrades itself to equal mode and another controller in equal mode becomes master. This does not have a major impact on load balance, but can be very helpful when we need to update controllers, deal with failures quickly, or even security applications.

To illustrate the need for and the applications of these new protocols, let us consider two switches S_1 and S_2, as well as two controller instances C_1 and C_2. We assume C_1 and C_2 are in equal and slave mode for switch S_1, and in master and slave mode for S_2. If our goal is to reduce the load on C_1, we can use traditional load migration protocols that change the master and slave associated with S_2 (i.e. C_1 will become a new slave, and C_2 will be the new master for S_2). However, that would lead to interruptions in network operations as the master controller will be paused during the handover process.

Alternatively, we can achieve the same goal of reducing the load on C_1 by switching C_1 to an slave for S_1 and turning C_2 to an equal. This will not lead to any interruptions as the master for S_1 remains operational. This is the main advantage of Protocol 2 (illustrated in Figure 2) over previously known load migration protocols.

III. Experiments

In this section, we simulate the scenario described in the previous section to compare the performance of our new proposed protocol with existing protocols ERC [3] and 4-phase [2]. We measure the handover time from the time that the start signal is issued, until the time that the initial slave is ready to start processing messages buffered during migration. Since none of the previous protocols can handle the required role change directly, we make some modifications to provide an apple-to-apple comparison of the protocols. For the ERC protocol, we can simply interchange the "initial master" (source controller) role with an equal in the protocol's description. However, for the "final master" in that protocol description, the only required change is to send "equal" `ROLE_REQUEST` rather than "master" request. For the 4-phase protocol, we make similar role changes in its fourth phase, and completely ignore the first phase.

The simulation environment is implemented using Mininet and we use D-ITG as our traffic generator. We assume that each switch generates the same amount of load, and vary the load on C_1 by increasing the number of switches associated with it (denoted by n), from 10 to 50 switches. There is always a single switch connected to C_2. We repeat each experiment 20 times and report the average running time of each setting.

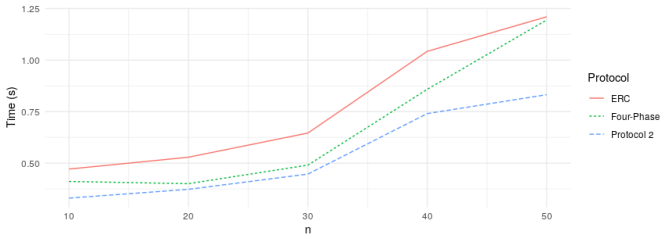

Fig. 3. Running-time comparison of migration protocols under different loads.

As expected, by increasing the load on C_1, we see an increase in the total running time of all the protocols. Having said that, the newly proposed protocol consistently outperforms the 4-phase and ERC protocols by an average of 20% and 31%, respectively, with higher improvements at higher loads.

References

[1] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, and W. Snow, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. New York, NY, USA: ACM, 2014, pp. 1–6.

[2] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; an elastic distributed SDN controller," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. New York, NY, USA: IEEE, 2014, pp. 17–27.

[3] M. A. Beiruti and Y. Ganjali, "Load migration in distributed sdn controllers," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.