Load Migration Protocol for SDN Controllers

Mohammad Amin Beiruti, Yashar Ganjali
Department of Computer Science, University of Toronto

***Abstract*—The dynamic nature of network traffic can lead to load imbalance amongst controller instances in a distributed SDN controller. A highly loaded controller instance can be slower in responding to datapath queries, and can slow down the entire control platform. In this poster, we present a new and efficient load migration protocol for shifting input load associated with overloaded controller instances towards lightly loaded instances. Unlike existing protocols for load migration, our protocol ensures consistency among controller instances, and can handle failures during migration procedure. Our protocol reduces the migration time by 20-55%, and the migration buffer size by 10-15%.**

I. INTRODUCTION

Modern SDN solutions rely on distributed controllers to enhance resiliency to failures, and to decrease response times in large-scale networks. In any distributed controller, different controller instances can have unbalanced load due to natural difference in arrival rates of switches in various topological layers, as well as fluctuations in network traffic. The natural solution to the load imbalance problem is to shift some of the input load associated with overloaded controller instances towards lightly loaded controller instances.

The first protocol for switch migration between controller instances was introduced by Dixit et al. [1]. They proposed a 4-phase protocol to hand over a given switch between two controllers instances. This protocol relied on OpenFlow [2] as the API between controllers and datapath elements, and was later used as a primitive for load balancing, power saving, and resource allocation in several solutions.

Unfortunately, this 4-phase protocol has several shortcomings. First, it does not cope with failures during the handover period. Second, the 4-phase protocol focuses only on two controller instances (source and destination of migration which are in master and slave modes respectively) rather than the entire system as a whole. During migration, there might be controller instances in *equal* mode that are supposed to have identical states to the master controller (see OpenFlow 1.5 specifications [2]). By ignoring these equal controller instances, the 4-phase protocol leads to inconsistent views in the equal-mode controller instances. The problem becomes worse in the presence of failures, as one of these equal-mode controller instances might be elected as the new master.

In this poster, we introduce a new 3-phase protocol for load migration in distributed SDN controllers. Unlike the previous 4-phase protocol, our protocol is resilient to failures, and ensures all other controller instances in equal mode are kept in a consistent state. Our design leads to 20-55% reduction in migration times, and 10-15% improvement in buffer size requirements.

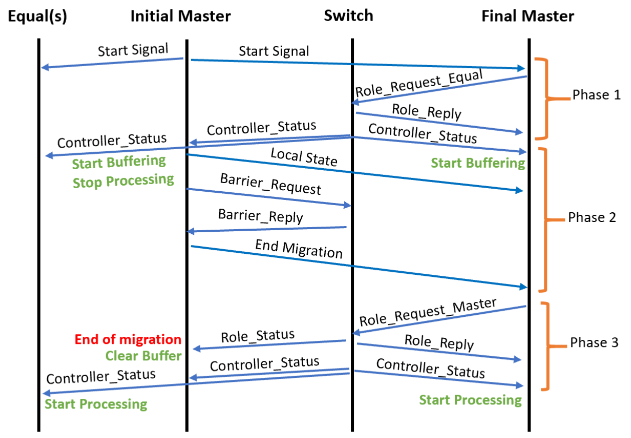

Fig. 1. Newly proposed 3-phase protocol.

Control State in SDN. The control plane is expected to keep the *state* of various *network applications*. The details of how this state is preserved depends on the design of each SDN controller platform, but regardless of these details, the control state can be categorized as follows. Each SDN switch contains some control state called ***switch state*** accessible by controller instances (e.g. packet/byte counters). Depending on the control platform's design, ***network controller application state*** might be stored in a simple key-value store or a shared storage system. The control platform ensures reliability and consistency of this storage. Each controller instance has some *ephemeral* ***local control state*** not managed by the control plane. If the controller instance fails, this state is lost. We need to ensure this state is transferred to a new controller during migration.

II. DESIRED PROPERTIES

In order to ensure the control plane does not end up with a corrupt view of the network state during the migration process, there are several properties that we need to uphold:

Safety. Every asynchronous message should be responded by at most one controller.

Serializability. Controller instances should process messages of each switch in the same order they are received by the control plane.

Liveness. Each switch has to have one active master controller at all times.

Failure Resiliency. If the destination controller instance fails during migration, the initial master instance should be able to resume normal operation without losing state.

Consistency. The master controller's view of a specific switch should be compatible with the view of all equal controllers.

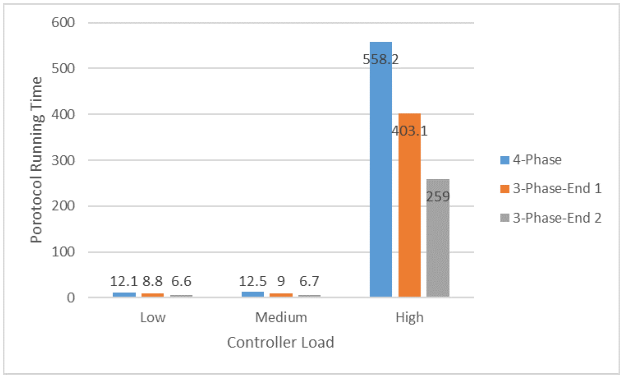

Fig. 2. Migration run-time.

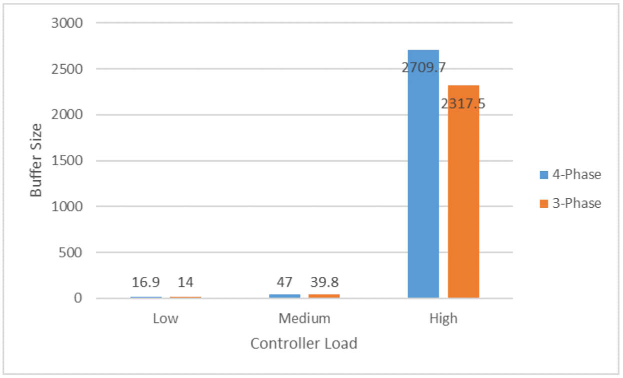

Fig. 3. Migration buffer size.

This is to ensure equal mode controllers can take responsibility of the switch during overload, or failures.

III. New 3-Phase Protocol

The previously known 4-phase protocol [1] only has safety, serializability, and liveness properties and does not support failure resiliency, and consistency. Without these properties, the state of any controller instance in equal mode might not match that of the master controller, and the 4-phase protocol cannot handle failures during the migration process.

Figure 1 shows our new 3-phase protocol for shifting the load of a given switch from its master controller instance to a slave. Unlike the 4-phase protocol, the initial master controller buffers incoming messages during the migration in order to ensure the state of the controller can be recovered in case of failures. This buffer also serves as a coordination mechanism between two controller instances, which is why we can eliminate a complete phase in our proposal. Also, unlike the 4-phase protocol, we explicitly deal with controllers in equal mode and ensure their state is kept up-to-date during the migration process.

Phase 1. The initial master controller initiates the migration. The final master requests a role change from slave to equal. After processing the role request message, the switch sends a Role_Reply message back to the final master.

Phase 2. After the initial master receives the Controller_Status packet, it starts buffering any messages from the switch. The initial master sends a copy of its local state to the final master. Any changes in state beyond this point is replicated on the final master. The initial master sends a Barrier_Request to make sure the switch does not have any pending requests. When the initial master receives the Barrier_Reply from the switch, it can safely signal the final master to hand over the responsibility.

Phase 3. Once the final master controller receives the end of migration signal and after processing the local state messages, it sends a Role_Request message to the switch. The switch changes the role of the current (initial) master to slave (and informs it with a Role_Status message). It also grants the master role to the final master controller instance. The switch broadcasts a Controller_Status message to all its corresponding controllers including equal and slave mode controller instances.

Desired Properties. We can formally prove that the 3-phase protocol supports all five desired properties. The details are omitted due to space limitations.

IV. Evaluation

We designed and implemented an object-oriented distributed SDN simulation environment with OpenFlow support for evaluating our protocol. The blue and orange bars in Figure 2 show the running time of the 4-phase and 3-phase protocols. The new protocol has a 20-32% shorter running time, despite the fact that the 4-phase protocol misses failure resiliency and consistency properties. The gray bar in Figure 2 represents a simplified version of the 3-phase protocol that does not support resiliency and consistency (to provide apple-to-apple comparison). The simplified protocol runs in less than 50% of the time for the 4-phase protocol. Interestingly, the improvement in running time is higher under higher load scenarios.

Figure 3 shows that the new protocol requires 10-15% less buffering compared to the 4-phase protocol. This is due to the fact that the 3-phase protocol is completed faster, and therefore, the migration buffer has less time to grow.

V. Conclusion and Future Work

In this poster, we presented a new 3-phase protocol for load migration in distributed SDN controllers. Our protocol is faster and requires less memory compared to the existing 4-phase protocol. Possible directions for the future of this work include designing protocols for other combinations of master, equal, and slave mode instances, as well as techniques to combine these protocols to transform a controller plane to a desired state (e.g. for load balancing, or resource optimization).

References

[1] 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.

[2] A. Nygren, B. Pfaff, B. Lantz, B. Heller, C. Barker, C. Beckmann, D. Cohn, D. Malek, D. Talayco, and D. Erickson, "Openflow switch specification version 1.5. 1," *Open Networking Foundation, Tech. Rep.*, 2015.