A Self-Organizing Distributed and In-Band SDN Control Plane

Marco Canini* Iosif Salem‡ Liron Schiff† Elad M. Schiller‡ Stefan Schmid§
*Université catholique de Louvain ‡Chalmers University of Technology
†Tel Aviv University §Aalborg University & TU Berlin

***Abstract*—Adopting distributed control planes is critical towards ensuring high availability and fault-tolerance of dependable Software-Defined Networks (SDNs). However, designing and bootstrapping a distributed SDN control plane is a challenging task, especially if to be done in-band, without a dedicated control network, and without relying on legacy networking protocols. One of the most appealing and powerful notions of fault-tolerance is self-organization and this paper discusses the possibility of self-organizing algorithms for in-band control planes.**

I. Our Contribution

Computer networks have become a critical infrastructure of our society. However, traditional computer networks are often inflexible, complex and error-prone, raising concerns regarding their dependability. Over the last years, Software-Defined Networks (SDNs) have emerged, promising interesting opportunities for designing more dependable networks [1].

SDNs advocate for software-based control of a network based on a simple, logically-centralized global network view. While the literature has well articulated the benefits of the separation between control and data planes [2], the question of how connectivity between these two planes is maintained – that is, paths for supporting switch-controller and controller-to-controller communications – has not received much attention.

The problem is further amplified by the fact that the logically-centralized SDN control plane actually means physically distributed, for a plethora of reasons, including reliability, availability, scalability, and latency (cf. [3] and references therein). In practice, most SDN deployments resort to out-of-band control, where control plane packets are carried by a dedicated management network. The management network runs its own routing system, which typically is realized using traditional routing protocols such as STP or OSPF.

Instead, in-band control is desirable for many reasons including its economical benefits (in certain contexts, such as carrier networks, out-of-band control would be prohibitively expensive). In essence, with in-band control there is no reason to build, operate, and ensure the reliability of a separate network.

This work contributes an important module for many fault-tolerant distributed systems: a self-organizing network control plane. In particular, we find that SDNs offer interesting opportunities for designing robust and highly available network structures. We sketch two approaches for implementing a self-organizing SDNs.

II. System Settings

We consider a network connecting two kinds of components, *controllers* and *switches*, henceforth also called *nodes*. The controller nodes compute, communicate and coordinate the network data flows. The switches are nodes that relay the packets of the data plane, communicate with the controllers, and have unique addresses. Switches are basic and "passive" devices: they cannot perform general purpose computations, because they have to follow a set of rules that the controllers define for them. The term switch configuration refers to the set of match-action rules with priorities and (possibly infinite) timeout values. The configuration can be changed either (1) by a controller or (2) through a timeout (which upon firing, causes the corresponding rule to be deleted). We consider two configuration settings: one that is *timeout-based*, which uses rules with (finite) timeouts, and another that is *timeout-free*, which does not use any (finite) timeout. A switch can be in one of two possible states: *managed* by one or more controllers or *unmanaged*, say, because the timeout of its management rule has expired. We consider two variations of the manner in which a switch becomes managed. In the first way, the switch takes the initiative to contact a given controller. In the other way, the initiative is on the controller side.

III. Our Approaches

The SDN control plane, that is, a collection of network-attached servers, must have connectivity to the data plane. Self-organization is a natural approach to meet this goal while coping with dynamic conditions, such as arrival and departure of controllers [4], arbitrary topology changes (switch or link failures), and communication errors (packet losses or delays). However, the design of self-organizing mechanisms to build and maintain connectivity between a distributed control plane and the data plane raises several fundamental challenges.

Our goal is to place each switch in the network under the control of a controller and establish routes that support connectivity of the control plane to the switches as well as among the controllers themselves. We propose the following two approaches.

A Timeout-based Approach: *Each switch opens an OpenFlow connection to a single controller whose address is preconfigured in the form of a virtual anycast address. Every controller constructs an (in-band control) tree. This tree reaches all switches that this controller manages as well as the other controllers in the network, as in [5].*

To allow the system to add and remove controllers and switches, we aim to minimize the prior knowledge that any

Algorithm 1: Self-organizing SDN, code for controller p_i.

```
1  local state:
2  responses, most recently received query responses;
3  do forever begin
4      make sure that p_i's direct neighborhood appears in
         responses and remove from responses stale
         information, e.g., packet forwarding rules that
         consider nodes that are unreachable according to
         responses;
5      foreach node p_j reachable from p_i do
6          query p_j's configuration and local neighborhood
7      foreach switch p_j reachable from p_i do
8          p_i becomes an equal manager of p_j and updates
             its forwarding rules so that p_i can communicate
             with p_j (and any other controller);
9  upon arrival of p_j's query response m begin
10     if no space to store m then responses ← ∅;
11     store m in responses (remove p_j's old responses)
```

switch needs to have about the controllers. We use a pre-configured *anycast controller address* at the switches: a logical IP address shared by *all controllers*. When unmanaged, a switch periodically attempts to connect with any controller using the pre-configured remote controller address, i.e., the virtual anycast address. When a switch is managed by a controller, that controller can access and modify the configuration of the switch that determines its behavior.

We devise a control plane in which the controllers are connected among themselves. This allows the controllers to use in-band connections for the sake of, for example, deciding which controller should be the master (unique) manager of an individual switch. Each controller p_c attempts to manage exclusively, i.e., in the master mode, a set of switches by preconfiguring them to accept connections from p_c. Note that a controller can take control of an unmanaged switch using an atomic operation [6]. This way, we provide an in-band management that decides which controller should be the manager of a switch following a "first-come-first-served" approach.

Our approach allows every switch to notice when it is unmanaged, e.g., when its controller is not live and connected. The key difficulty here is the passive nature of the switches. To overcome this difficulty, the algorithm need to make sure that whenever a switch is disconnected from its controller, the switch detects controller inactivity and transitions to the unmanaged state. In turn, this allows other controllers to connect and re-establish the in-band control plane. We address this challenge through *rule timeouts* to detect inactivity and a set of *a-priori rules* that encode the switch behavior in its unmanaged state. These rules are low priority rules that are masked by other rules, which are installed (and regularly refreshed) by a controller while the switch is in its managed state. Importantly, we must be careful to define this unmanaged configuration through a set of a-priori rules that ensure that some controller can eventually learn and reestablish connectivity of an unmanaged switch.

One of the important aims is to establish in-band OpenFlow connections from any switch to a controller. To allow connecting between controllers and switches to which there is no direct connection, the algorithm lets each controller install rules on the switches that it is directly connected to [5]. Then, the algorithm lets the controller install rules on switches that are two hops away from it and so on. This iterative process expands the controller's (reachable) network. We note that our approach can also ensure that the controller installs in-band rules in a way that does not conflict with the rules for regular data plane traffic.

A Timeout-free Approach: *All controllers discover the network topology of the SDN while letting each controller become the (equal) manager of every switch. Each controller is the root of a tree that spans, in an in-band manner, the entire control network (Algorithm 1). Once a controller has discovered the entire network, it can remove stale information.*

In this approach, we avoid preconfiguring addresses, such as the anycast address that we use in the first approach. This allows us to deal with the corruption of these addresses and stop relying on a network infrastructure that supports anycast addresses. Algorithm 1 configures the switches in a way that allows (1) each switch to participate in the discovery of its local topology and (2) any controller to open an OpenFlow session to them (in an equal mode). By that, the algorithm avoids preconfiguring any address, which could be subject to errors. The controllers run a topology discovery algorithm that calls for the discovery of the local topology (lines 4 to 6 and 9 to 11) and installs packet forwarding rules that facilitate a connection between every switch and controllers (lines 7 and 8). By coupling, in a step by step manner, the local topology discovery and the construction of paths from every controller to every switch, the algorithm can include in the control network more and more switches. We propose to extend Algorithm 1 as follows. Once a controller has discovered the entire network topology, it can also remove any information on the switches, such as forwarding rules, which was installed by any controller that does not appear in the discovered topology. This addition will let any controller clean up the switch configuration after the crash of any other controller.

Acknowledgements. This work was partially supported by Swedish Vinnova's project *C-ITS Testplattform för framtidens transportsystem (CHRONOS steg1)* as well as by the Danish Villum project *ReNet*.

REFERENCES

[1] M. Canini, D. Venzano, P. Peresíni, D. Kostic, and J. Rexford, "A NICE Way to Test OpenFlow Applications," in *NSDI*, 2012.

[2] N. Feamster, J. Rexford, and E. W. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, Apr. 2014.

[3] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *HotSDN*, 2012.

[4] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," in *HotSDN*, 2013.

[5] L. Schiff, S. Schmid, and M. Canini, "Ground Control to Major Faults: Towards a Fault Tolerant and Adaptive SDN Control Network," in *DISN*, 2016.

[6] L. Schiff, S. Schmid, and P. Kuznetsov, "In-Band Synchronization for Distributed SDN Control Planes," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, Jan. 2016.