CORONET: Fault Tolerance for Software Defined Networks

Hyojoon Kim
Georgia Institute of Technology
joonk@gatech.edu

Jose Renato Santos
HP Labs, Palo Alto
joserenato.santos@hp.com

Yoshio Turner
HP Labs, Palo Alto
yoshio_turner@hp.com

Mike Schlansker
HP Labs, Palo Alto
mike.schlansker@hp.com

Jean Tourrilhes
HP Labs, Palo Alto
jean.tourrilhes@hp.com

Nick Feamster
University of Maryland – College Park
feamster@cs.umd.edu

***Abstract*—Software Defined Networking, or SDN, based networks are being deployed not only in testbed networks, but also in production networks. Although fault-tolerance is one of the most desirable properties in production networks, there are not much study in providing fault-tolerance to SDN-based networks. The goal of this work is to develop a fault tolerant SDN architecture that can rapidly recover from faults and scale to large network sizes. This paper presents CORONET, a SDN fault-tolerant system that recovers from multiple link failures in the data plane. We describe a prototype implementation based on NOX that demonstrates fault recovery for emulated topologies using Mininet. We also discuss possible extensions to handle control plane and controller faults.**

I. Introduction

The Software Defined Networking paradigm advocates using a logically centralized controller that makes traffic forwarding decision while local switches are responsible for performing packet forwarding in the data plane, thus separating the control plane from the physical data plane. SDN-based networks are increasingly been deployed in testbed as well as production networks. For example, Google presented its deployment of OpenFlow, the most popular protocol for SDN, on Google's intercontinental WAN network at the OpenFlow Networking summit in April, 2012 [7].

Although the community has explored a large space about what kind of new functionalities SDN can deliver, there has been little discussion on how to deal with an age-old yet common problem in computer networks: *network failures*. Network failure, such as network device failures of link disconnections[1], can obstruct normal traffic forwarding even if there is an alternate path in the network. However, the field lacks sufficient research and experience in building fault-tolerant SDN-based networks. Maulik Desai *et al.* explored how to cope with link failures in SDN-based networks, however the system does not completely recover from faults [1]. PortLand [5] does recover from link failures, but is restricted to multi-rooted tree topologies.

There are three fault domains in SDN-based networks. (1) *Data plane*, where a switch or link fails, (2) *control plane*, where the connection between the controller and switch fails, and (3) *controller*, where the controller machine fails. This work plans to address these three fault domains but has initially focused on building a fault-tolerant SDN-based solution for *data plane faults*. The main challenges on building a fault-tolerant system based on SDN are:

- **Existing solutions from legacy networks do not work out-of-the-box in SDN networks.** Running existing solutions that normally operate on legacy networks, such as *rapid spanning tree protocol*, alongside a SDN controller application do not work properly without careful planning or extensive modification. This is primarily because two control planes coexist yet are decoupled from each other, without any interaction or information exchange between them.
- **Despite of advantages, pure SDN-based solutions are not efficient.** In SDN-based networks, the central controller can collect information from each network element, possessing a global knowledge of the network. With global information, the controller can run centralized algorithms that are potentially more efficient than distributed algorithms that have limited information in case of failures. However, OpenFlow SDN is a flow-based solution where potentially every first packet has to consult the controller for forwarding decision. This does not scale if the number of hosts increases because the number of unique flows can grow non-linearly. As a consequence, the number of forwarding rules that need to be modified to recover from a failure can be very large and prolong recovery time.

II. CORONET: Fault Tolerance for SDN

CORONET, COntroller based RObust NETwork, is a scalable and efficient fault tolerant system with multipath support. CORONET has the following properties:

- **Fast recovery.** CORONET recovers from switch/link failures in a sub-second timescale after it detects a fault.
- **Scalable to large networks.** As network size increases, the number of flow table entries or number of control messages remains manageable.

[1]Network failures are caused by many reasons, *e.g.*, a software bug in network devices or misconfiguration by operators. This paper limits the cause to be device failures or link disconnections in the network.

978-1-4673-2447-2/12/$31.00

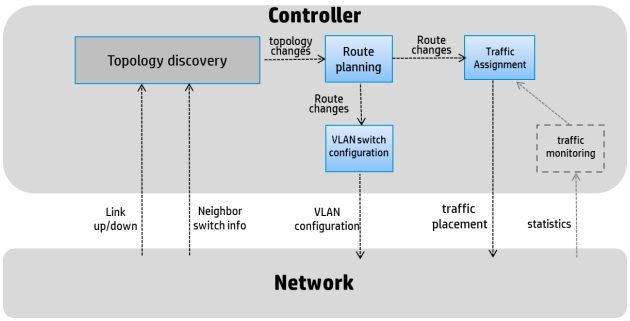

Fig. 1: CORONET Architecture.

- **Multipath routing.** CORONET makes use of multiple alternate routing paths in the network.
- **Works with arbitrary networks.** CORONET works with any network topology, including fat tree, clique, and mesh topologies.
- **Single control plane.** The central controller is the only entity that makes forwarding decisions.

CORONET runs an efficient centralized algorithm in the controller with global knowledge of the up-to-date network status, as opposed to distributed algorithms that have limited information. CORONET uses local switch mechanisms for tasks that do not require global knowledge. For example, packet forwarding is primarily performed by VLAN mechanism implemented in local switches. Failure detection and topology discovery also relies on local switch mechanisms such as Link Layer Discovery Protocol, or LLDP. Hence, CORONET is able to support full data forwarding rate with minimized control traffic with fast failure detection. CORONET uses packet classification at edge switches to map traffic to VLANs, which can be programmed using OpenFlow API.

A. Architecture

Figure 1 shows the architecture of CORONET. The controller comprises four modules that are responsible for performing specific tasks.

The *Topology discovery* module periodically collects topology information and receives asynchronous events upon link/switch failure. The module ensures that CORONET has up-to-date information of network topology status. The *Route planning* module calculates multiple routing paths based on the topology information. Routing paths are computed by the *VLAN growing algorithm*, which creates multiple link-disjoint shortest routing paths using Dijkstra's shortest path algorithm. The link-disjoint property provides better reliability by minimizing the number of affected routing paths when a link fails. The *VLAN switch configuration* module configures multiple switch ports with relevant VLAN IDs to enforce each routing path. The *Traffic assignment* module assigns host traffic to routing paths. Currently, the algorithm assigns host traffic to a routing path (VLAN ID, in this case) randomly or in a round-robin fashion. However, we envision to incorporate a separate *traffic monitoring* module (as shown as a dotted box in Figure 1), which provides traffic statistics feedback so that the module can perform dynamic load balancing.

B. Implementation

The CORONET controller prototype is built on top of NOX [2], a platform that provides APIs for SDN applications to interact with OpenFlow switches. CORONET is compatible with OpenFlow specification version 1.0.0 [6]. We evaluate our CORONET prototype using Mininet, a virtual network topology emulator [4], which can generate customized virtual network topologies in a Linux machine.

III. Future Work

Building a general framework: CORONET's use of VLAN dramatically simplifies packet forwarding, reducing the number of forwarding rules and thus improving scalability compared with a standard Openflow approach. SDN applications in CORONET can only specify logical paths implemented by VLANs rather than specifying physical paths directly. While many existing Openflow applications directly control the path of packets, we believe that these applications could be rewritten using the CORONET framework. As part of future work, we plan to evaluate the generality of CORONET framework to support common SDN applications, and build a general framework that allows seamless integration with any SDN application.

Control plane and controller reliability: Currently, CORONET only supports fault-tolerance for data plane failures. Our work envisions to provide a complete fault-tolerant solution that recovers from multiple failures coming from other fault domains in SDN-based networks. For controller faults we plan to investigate and compare two approaches: **1)** an approach based on a distributed hash table inspired by Onix [3] and **2)** a traditional software failover solution based on heart beat detection. For the control plane we plan to investigate the feasibility of using traditional distributed mechanisms such as the rapid spanning tree protocol, and compare with a controller based approach in which the controller reconfigures the control plane when faults are detected.

References

[1] M. Desai and T. Nandagopal. Coping with link failures in centralized control plane architectures. In *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, pages 1 –10, jan. 2010.

[2] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.

[3] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[4] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: Rapid prototyping for software-defined networks (at scale!). In *Proc. HotNets*, Oct. 2010.

[5] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer2 data center network fabric. In *Proc. ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[6] OpenFlow Specification v1.0. http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf.

[7] OpenFlow Networking Summit. http://opennetsummit.org/, Apr. 2012.