When Raft Meets SDN: How to Elect a Leader and Reach Consensus in an Unruly Network

Yang Zhang, Eman Ramadan, Hesham Mekky, Zhi-Li Zhang
University of Minnesota
Introduction

Consensus Algorithm
Introduction

Consensus Algorithm

Consensus algorithm is essential for SDN distributed control plane
Problems in SDN Distributed Control Plane

Cyclic dependencies
- control network connectivity
- consensus algorithm
- control logic managing the network

In consensus algorithm
- server failure has been studied for decades
- full mesh connectivity is assumed
- what if network fails?
- new failure scenarios arise in SDN
RAFT: a representative consensus algorithm

• At any given time, each server is either:
  – Leader: handles all client interactions, log replication, etc.
  – Follower: completely passive
  – Candidate: used to elect a new leader

• Normal operation: 1 leader, N-1 followers
RAFT Leader Election

Vote criteria: 1) **highest term**, 2) **latest log**
*Term is defined as virtual time period in Raft*
RAFT MEETS SDN

Control Cluster under **Normal Operations**.
Control Cluster under **Normal Operations**.

Oscillating Leadership. **Condition.** Up-to-date servers have a quorum, but they cannot communicate with each other.
RAFT MEETS SDN

No Leader Exists.

Condition. Some servers have a quorum, but they have obsolete logs, and servers having up-to-date logs, do not have a quorum.
POSSIBLE SOLUTIONS

• Solution Expectation
  • all-to-all connectivity among cluster members as long as the network is not partitioned.
• Gossiping (overlay network)
  • Pros: easy to implement
  • Cons: no guarantee to work in all scenarios; heavy overhead
• Routing via Preorders
  • Pros: built-in resiliency in control plane; no modification to consensus algorithm
  • Cons: requires path calculation ahead of time
Routing via Preorder: Failure Handling

• Failure Handling Process:
  • Upon failures, use alternative outgoing links if exist
  • Group table is used for implementing all possible alternative paths

It guarantees a network where two nodes are always reachable as long as there is no partition.
PRELIMINARY RESULTS

• Experiment Setup

  • Raft Implementation: Raft C++ implementation in LogCabin
  • Six Docker containers: 5 servers and 1 client
  • Five Software switches: Open vSwitch
  • Simulating the two failure scenarios:
    ▪ Oscillating Leadership
    ▪ No Existing Leader
Raft: leadership keeps oscillating among servers (unstable).

Raft: no viable leader (liveness lost).

PrOG: leadership is stable.

Vanilla Raft is not stable under failure scenarios, while PrOG-assisted Raft is stable.
Client suffers much more failed attempts for accessing cluster leader in vanilla Raft.

Latency of a request operation increases under failure scenarios
Summary

• SDN controller liveness depends on all-to-all message delivery between cluster servers
• Raft is used to illustrate the problem induced by interdependency in the design of SDN distributed control plane
• Possible solutions are discussed to circumvent interdependency issues.
• Preliminary results show the effectiveness of PrOG in improving the availability of leadership in Raft used by critical applications like ONOS.