On the design of practical fault-tolerant SDN controllers

Fábio Botelho, Alysson Bessani, **Fernando M. V. Ramos**, Paulo Ferreira
No news in this slide

- SDN is a success story
  - Google’s B4 & VMware’s NVP
  - Nicira acquisition by VMware
  - Industry-led consortia as ONF or OpenDaylight (ODL)
  - EWSDN, HotSDN, and others

- No matter what happens tomorrow, SDN will certainly appear as one of the highlights of the *encyclopaedia of networking*. 😊
With success comes responsibility

From the outset, there is a need to consider

- scalability and performance
  - already a good amount of work here, both research- and industry-wise
- availability
  - also some work here, particularly in industry (not so much in research)
  - **Our target**
- resilience (security and dependability)
  - increasing concern

in the design of SDN infrastructure.
Fault tolerance in SDN

• We look at controller fault tolerance as a means to achieve high availability
• Fault tolerance in SDN (as per [CORONET])
  – Data plane (switch or link failures)
    • Some work here ([CORONET][FatTire])
  – Control plane (failure in switch-controller connection)
    • Recent Openflow standard versions offer help here
  – Controller
    • Some work on distributed controllers, although the emphasis is usually on performance and scalability
    • Highlights:
      – [ONIX] is a distributed closed-source controller (core element of both B4 and NVP) concerned with scalability, availability and generality
        » Highly successful, but complex, and closed
      – [ONOS] is a recent effort (HotSDN’14) for an open-source distributed controller
      – ODL also concerned with fault-tolerance, but work still in its infancy
What I have to offer today

• **A discussion** on the design and implementation of practical fault-tolerant SDN architectures

• The Onix paper provided a first insight
  – Our proposal is a *zillion times* more modest
  – But is also more focused (and therefore more detailed) on the *fault-tolerance* aspect
  – we hope such discussion to offer food for thought in the design of new SDN controllers

• Did I already say that our fault-tolerant design is modest?
  – A *centralized* controller with one (or more) backup(s)
  – for *small to medium-size* networks
  – *fault-tolerant*, with some *nice properties* (*safety, liveness*), and assuring a *smooth* transition in case of controller failures
So let’s start from the beginning

- The **construction** of a fault-tolerant (FT) SDN architecture
- The design & implementation of our **SMaRtLight** architecture
- A quick **evaluation**, and
- A **discussion** to wrap-up
We start with a centralized controller

Recall: this is bad, single Point of Failure (PoF)!
and now we add a backup controller

Wait, the replicas need to coordinate!
OK, so we run coordination protocols between the replicas
(leader election, fault detection)

Another problem: on failure the replica starts with an empty state!
Possible solution: use a shared data store to store the NIB

Wait: now the data store is a single PoF!
OK, so use a replicated, fault-tolerant shared data store

OK, no single PoF now. Nice.
• The design & implementation of our SMaRtLight architecture
Similar to the previous design

Our FT shared data store is implemented as a Replicated State Machine: https://code.google.com/p/bft-smart/

With 3 slight (but important) changes
#1: Remove coordination between controllers

Simplify controller design by leveraging on the shared data store
#2: Add small coordination module to the controller
Coordination module: some detail

• This module runs an algorithm that periodically calls the `acquireLease(id, L)` operation in the data store
  – This function returns the id of the primary
    • If there is no primary (there was a fault in the primary), then the invoker becomes the new primary
    • If the invoker is the primary, its lease its renewed
  – The primary interacts with the switches or data store only if the predicate `iAmPrimary()` is true.
  – This module thus implements both leader election and fault detection

• Properties enforced by this module
  – Safety: at any point in the execution, there is at most one primary controller
  – Liveness: Eventually some correct process will become a primary

• Check paper for a full description of the algorithms, and its extended version in arXiv for proofs for these properties
#3: Add a cache to the controller to improve performance on reads
We anticipate our architecture to make efficient use of the cache

• We expect the size of the NIB not to be significant when compared with the amount of main memory
  – The [Onix] paper suggests a single Onix instance (on a server-grade machine) can easily handle networks of millions of entities
    • They even show that a mere 2GB is enough for 1m entities with 30 attributes each
  – So all reads can be absorbed locally
• Our solution is centralized, so there is no need for complex, low-performant cache invalidation protocols
• So let's start from the beginning

  • The construction of a fault-tolerant (FT) SDN architecture
  • The design & implementation of our SMaRtLight architecture
  • A quick evaluation, and
  • a discussion to wrap up
We aim to answer two questions

• How does the introduction of a fault-tolerant data store affects the performance of a centralized controller?

• What is the performance impact of controller faults?
Experiments setting

- **CBench** simulating switches sending *packet-in* messages to the controller
  - This stresses the controller, but we expect the data store to be the *bottleneck*
  - Hence, we added a dummy application that processes requests locally with a probability $P$ (cache hit) accessing the data store with a probability $1-P$ (cache miss)
    - payload of 44 bytes (learning switch application)
    - we vary $P$ to simulate different applications
- We use 2 controller replicas and 3 data store replicas in all experiments
  - Each replica runs a quad-core 2.27 GHz Intel Xeon machines with 32 GB RAM, interconnected with GE.
• Of course, adding fault tolerance has a cost. Let’s put it into perspective
  – Floodlight processes 2.5Mflows/s, so we’re roughly one order of magnitude below
    • (BTW, we are pretty sure we can perform a bit better, as we’re still under 70% the expected capacity of our data store.)
  – In Floodlight a synchronous update (to a disk log for recovery after a failure) would reduce throughput to 200 flows/s (two to three orders of magnitude worse)
Throughput in the presence of faults

- Backup takes over after less than 1 second, and it takes around 4 seconds for the throughput to return to previous levels
  - This is the time it takes for switches to be informed and to start sending traffic to the new primary controller, and for it to warm-up
- Note that the crash of a data store replica does not affect performance
So let's start from the beginning

- The construction of a fault-tolerant (FT) SDN architecture
- The design & implementation of our SMaRtLight architecture
- A quick evaluation, and
- a discussion to wrap-up
Rationale for our hybrid solution

• We opted for a hybrid replication solution:
  – passive (primary-backup) replication for controller replication
    • To preserve the familiar programming model of centralized controllers
  – active (state machine) replication for the data store
    • To avoid the well-know large reconfiguration time of primary-backup systems
Alternative #1

- We opted to develop a data store on top of a state machine replication library (BFT-SMaRt), but we could’ve used an alternative data store (e.g., Cassandra)
  - In that case we would need to integrate the coordination protocols with that data store (would be infeasible in some cases, non-trivial in others)
Alternative #2

• We could’ve used Zookeeper as coordination service between controllers and Cassandra as the shared data store
  – This would move complexity to the controller; our architecture is more modular
Alternative #3

• We could’ve used Zookeeper as the data store and coordination service as in our solution – we found it simpler to improve system performance by implementing specific operations in our custom data store that best fit the network application.
KEEP CALM ITS MY LAST SLIDE
A call of arms

We have been working on a comprehensive survey on SDN available in arXiv:


We intend to make it a "live document" using feedback from the community. For this purpose, we have set up a github page to receive feedback:

https://github.com/SDN-Survey/latex/wiki

Join us in this collaborative effort!
References

• [CORONET]  

• [FatTire]  

• [ONIX]  

• [ONOS]  
  – B. Lantz et al. ONOS: Towards an Open, Distributed SDN OS. In HotSDN ’14, 2014.