Delorean: Using Time Travel to Avoid Bugs and Failures in SDN Applications

Zhenyu Zhou†, Theophilus Benson†, Marco Canini*, Balakrishnan Chandrasekaran§
†Duke University, *KAUST, §TU Berlin

1. INTRODUCTION

Bugs are endemic in software and SDN applications (SDN-Apps) are no exception. SDN controllers are fragile: crashes induced by bugs in SDN-Apps can, in the worst case, cripple the entire SDN stack [2, 9]. Crashes of SDN-Apps result in service outages, which is simply a euphemism for loss of revenue. Businesses can lose, for instance, thousands of dollars per minute because of outages [1, 6–8]. We aim, hence, to minimize outages by tolerating SDN-App crashes, even in case of deterministic faults.

Although the SDN community has made progress in terms of designing better (or more robust) SDN-Apps, and in weeding out bugs through sophisticated testing methods, the question of *"How do you devise an online technique to safely and systematically circumvent a bug that manifests even in a well- designed and well-tested SDN-App running in a production environment?"* remains, largely, underexplored. LegoSDN [2] tackles the problem of SDN-App crashes head-on, offering a framework to rollback the effects of a crashed SDN-App and transform the crash-inducing input to handle deterministic faults. The system, however, *blindly* regards the last input as crash-inducing and treats SDN-Apps as black boxes. Orthogonal attempts to employ Paxos-style replication, e.g., Ravana [3] and Onix [4], lack support for tolerating deterministic faults.

We treat failures as *first-class citizens* and, in the event of an SDN-App failure, propose *Delorean* to provide a quick, safe, *online* recovery of the SDN-App. At a high-level, the recovery comprises a rollback of both control-plane and data-plane changes, and a transformation of the crash-inducing input—modification of an input event to one or more *semantically equivalent* but *syntactically different* input event(s)—to handle even deterministic faults.

Delorean employs *symbolic execution* (SE) to gain insight into the application logic. Through static analysis, Delorean discovers a set of possible *code paths* (i.e., path through the source code indicating the control flow for a given input) within the SDN-App along with a set of *path conditions* (i.e., predicates in a conditional statement, e.g., "if", that need to be satisified for the execution to proceed along a given code path). To counter scalability issues of SE, Delorean performs a one-time *offline* analysis of the SDN-App and persists the output, an *execution tree* comprising all code paths along with the corresponding path conditions, for later use in an online recovery.

Delorean aims to recover from both deterministic and non-deterministic bugs of SDN-Apps based on the insights provided by SE: Delorean attempts to identify the *crash path* (code path where the crash happened), and to steer the SDN-App away from the crash path.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author(s). Copyright is held by the owner/author(s).

SOSR '17, April 03 - 04, 2017, Santa Clara, CA, USA

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4947-5/17/04. . . $15.00

DOI: http://dx.doi.org/10.1145/3050220.3060610

2. SYSTEM DESIGN

Delorean is a *shim* layer between the SDN controller and SDN-Apps. The system architecture comprises six modules: *App Manager*; *Symbolic Execution Analyzer (SEA)*; *Crash Cause Analyzer (CCA)*; *Transaction Manager*; *Transformation Generator*; and *Recovery Optimizer (RO)*.

The App Manager records messages exchanged between the SDN-Apps and the controller. The SEA module symbolically executes the SDN-App code to build the execution tree. The Transaction Manager enables rollback of both control-plane and data-plane changes. The Transformation Generator generates transformations based either on heuristics or on predefined rules from the network operator.

Delorean addresses two fundamental challenges: (1) determine the precise input event in the past (referred to as a *rollback point*) to rollback the control plane and data plane to; and (2) determine the transformations to apply while minimizing both the recovery time and the likelihood of another crash (either during or after a recovery attempt).

2.1 Crash Cause Analyzer (CCA)

Determination of the rollback point requires striking a trade-off between efficiency (time required to complete recovery) and effectiveness (likelihood of a successful recovery): recovery becomes inefficient if we rollback too far, and ineffective if we do not go back far enough. The CCA module tackles this challenge with the insights gained from SE.

Given the the crash path and its path conditions, the CCA module iterates through the history of input events (and state-variable changes) of the crashed SDN-App, in reverse chrono-

logical order, to determine a crash cause. An input event is regarded as a crash cause if (1) it is *non-idempotent* and modifies one or more state-variables relevant to the path conditions of the crash path, and (2) prior to processing this input the execution of the SDN-App proceeded on a crash-free code path.

2.2 Recovery Optimizer (RO)

The objective of transforming an input event is simply to drive the execution of the SDN-App away from the crash path. Some transformations, however, may be "better" than others. In an effort to increase the likelihood of a successful recovery and minimize recovery time, the RO module consideres these "better" transformations before the others. To this end, the RO module ranks transformations using both domain-specific heuristics and empirical observations. For instance, the RO module analyzes each transformation to determine the *path overlap* between the code path of the transformed input event and that of the crash scenario. Path overlap is defined as the number of branches (involving conditionals) in the execution tree of an SDN-App that are common between two code paths. A long path overlap indicates a minimal change in the SDN-App behavior, i.e., on processing the transformed input the SDN-App deviates from the crash path at a (branch) location in close proximity to where the crash happened, while a short path overlap implies the contrary. Delorean ranks the transformations in decreasing order of path overlap in an effort to recover successfully while introducing only a minimal change.

3. METHODOLOGY

After an SDN-App crash Delorean proceeds as follows.
1. The CCA module determines the crash path, by analyzing the history of input events and changes in the SDN-App's states, and finds the root cause of crash.
2. The transformation generator produces a list of potential transformations of the crash-cause, which are then ranked by the RO module.
3. The transaction manager restarts the SDN-App and restores both the data-plane and control-plane states.
4. The recovery process terminates if the SDN-App processes the transformed events without crashing. Otherwise, the process resumes from *Step-3* with the next transformation in the list. After exhausting the transformations of the current crash cause, the system will revert to *Step-1* to find another crash cause from an earlier instance of time.

4. EVALUATION

For evaluations, we emulated the data plane using Mininet [5] and used custom Python scripts to setup the topology, generate traffic between hosts, inject failures, as well as collect measurements. Both Mininet and the scripts were run on a Linux (Ubuntu 14.04 LTS) server with 12 processor cores and 16 GB of memory. We used three different SDN-Apps—Learning Switch, Stateful Firewall and Forwarding Module.

Figure 1 compares a wide spectrum of rollback strategies. While LegoSDN always rolls back to the last input, Delorean

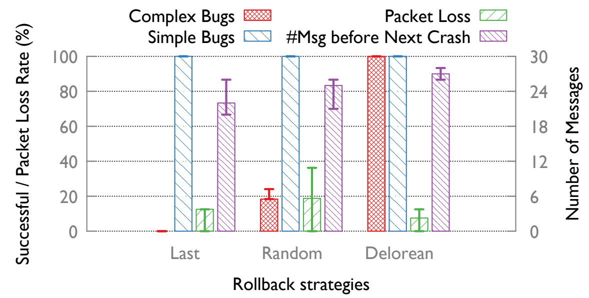

Figure 1: Comparison of different rollback strategies, showing that complex bugs require a more principled recovery strategy like that of Delorean

tries to determine an optimal solution in a principled manner. The *random* strategy uses an adhoc manner to find the optimal solution.

The results highlight that while a naïve rollback to the last event may suffice for simple failures, complex failures require a more principled approach. Unsurprisingly, randomly selecting the rollback point suffices in some cases. Figure 1 indicates that Delorean is better than LegoSDN [2] in efficiently recovering from SDN-App crashes. While not shown here, Delorean also performs better and faster than recovery strategies such as controller reboot or application reboot.

We also measured the time spent in the SE process, and observed that the time does increase with more code paths. We, however, note that Delorean only incurs the cost once per SDN-App; the result of this symbolically analysis is used until the SDN-App's source code is changed. The mean times were 2.33 s, 2.61 s and 2.78 s for Hub (1 path), Learning Switch (9 paths) and Hedera (13 paths), respectively.

In a nutshell, our preliminary results show that Delorean recovers successfully from a wide range of bug scenarios while related crash-recovery techniques falter in a majority of them.

5. REFERENCES

[1] Avaya. Network Downtime Results in Job, Revenue Loss. http://www.avaya.com/en/about-avaya/newsroom/news-releases/2014/pr-140305/, March 2014.
[2] B. Chandrasekaran, B. Tschaen, and T. Benson. Isolating and Tolerating SDN Application Failures with LegoSDN. In *SOSR*, 2016.
[3] N. Katta, H. Zhang, M. Freedman, and J. Rexford. Ravana: Controller Fault-tolerance in Software-defined Networking. In *SOSR*, 2015.
[4] 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 *OSDI*, 2010.
[5] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.
[6] A. Lerner. Gartner: The Cost of Downtime. http://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/, July 2014.
[7] Ponemon Institute. Cost of data center outages. *Data Center Performance Benchmark Series*, January 2016.
[8] C. Preimesberger. Unplanned IT Downtime Can Cost $5K Per Minute Report. http://www.eweek.com/c/a/IT-Infrastructure/Unplanned-IT-Downtime-Can-Cost-5K-Per-Minute-Report-549007, May 2011.
[9] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A Robust, Secure, and High-performance Network Operating System. In *CCS*, 2014.