Chaos Monkey: Increasing SDN Reliability through Systematic Network Destruction

Michael Alan Chang
Princeton University
machang@princeton.edu

Brendan Tschaen
Duke University
btschaen@cs.duke.edu

Theophilus Benson
Duke University
tbenson@cs.duke.edu

Laurent Vanbever
ETH Zürich
lvanbever@ethz.ch

1. INTRODUCTION

As modern networking applications become increasingly dynamic and high-bandwidth, software defined networking (SDN) has emerged as an agile, cost effective architecture with widespread adoption across industry. In SDN, the control-plane program runs on a logically-centralized controller which directly configures the packet-handling mechanisms in the underlying switches using an open API (e.g., OpenFlow). While the controller makes it exceptionally convenient for a network operator to control and manage a network, the controller requires complex logic and becomes a single point of failure within the network. As a result, configuration errors by the controller could be extremely costly for the network provider.

Several SDN controllers have been developed since the conception of SDN, and network operators have utilized very traditional means of identifying bugs in the controller, such as unit testing and model checking [1]. However, it has become apparent that these methods cannot practically handle the inherent complexity of the controller platform that manages large networks. Ultimately, one major source of this complexity are network failures, as they trigger execution of unexplored portions of code; these network failures are inevitable, costly, and considering all possible interleaving of bugs is simply unfeasible.

To address this problem, we propose "Chaos Monkey" a real-time post-deployment failure injection tool. Inspired by industry practices in the cloud [2], Chaos Monkey is intended to systematically introduce failure (e.g., link failure, network failure) into a network. Chaos Monkey is guided by the following design principles:

- **Realistic:** The distribution of the injected failures should mirror reality.
- **Manageable:** Minimize the changes of completely destroying the network.
- **Coverage:** The injected failures should maximize the amount of controller code executed.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790038

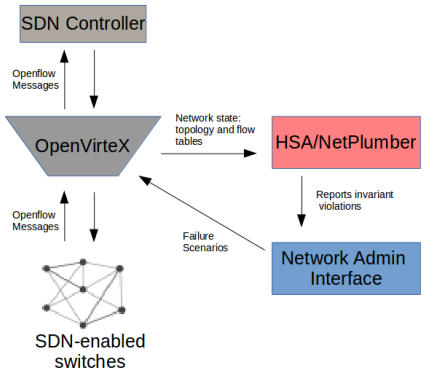

Figure 1: Overview of Chaos Monkey Architecture.

- **Leveraging network redundancies:** The injected failures should not disconnect the graph.
- **Detecting network-wide invariant violations:** The injected failures should aim at uncovering given invariant violations, e.g., reachability guarantees.

The concept of injecting failures into a live production network system is certainly counter-intuitive, but we assert that there is no better way to efficiently simulate actual network usage than within the actual runtime environment. Testbeds cover a limited case and simply do not scale up to production levels; some problems – inherently complex and difficult to reproduce – only manifest at scale.

2. IMPLEMENTATION

There are three major components that are necessary to achieve the ultimate goal of dynamic failure injection and invariant verification. These components and their roles are illustrated in Figure 1.

Chaos Monkey utilizes a OpenVirteX (OVX) [3] as a means of monitoring and affecting the network state. OVX sits between the network and the SDN controller, relaying messages between them. To perform the invariant checking, Chaos Monkey employs NetPlumber [4]. NetPlumber provides a real-time solution to verifying invariants, such as forwarding loops, in the network topology. This module receives updated flow table information from OVX, checks for invariants, and provides the network operator with feedback.

Next, we describe the Chaos Monkey Pipeline and how NetPlumber and OVX fit into it:

Step 1: Define Failure Scenarios The primary purpose of this module is to compute a set of devices to fail under constraints. The anticipated use case of Chaos Monkey is for the network operator to iteratively test and refine the way Chaos Monkey injects failure into the network. Ultimately, the nature of these failures are described by a failure model.

The Failure Scenario constrains the behavior of the Chaos Monkey and the definition of this model is critical in finding a balance where Chaos Monkey realistically injects as many effective failures as possible without totally disrupting the overall functionality of the network. Since there are often several possible sets of failure scenarios, the best models is governed by probabilistic models that can be toggled by the network operator. This allows the network operator to throttle the amount of failures during times of reduced network usage.

Step 2: Inject Failures Up to this point in the process, no failure has actually been injected into the network. The purpose of this module is to schedule and subsequently inject the failures. In the previous step, the network operator has specified which subset of network resources to target; this module determines how frequently Chaos Monkey will fail the nodes. For example, this can allow the network operator to schedule Chaos Monkey by throttling the amount of failures during times of increased network usage. As far as the actual injection of failure, Chaos Monkey has some selection of failure classes to choose from. The failures are injected into the network using OVX's REST API, which has been extended to accommodate additional failure classes. At this point in the research, there exist implementations for network failure scenarios (e.g. switch down, link down) and increasing latency in the OpenFlow channel. There are many failure scenarios to consider beyond these, and we are actively extending our API to encompass more failure scenarios.

Step 3: Check Network Invariants The final module consists of the network invariant checker. Once the failures have been computed and injected into the network, the invariant checker analyzes the latest updates. Upon receiving the new network state, such as a topology change or a new flow table entry, NetPlumber analyzes the information against its invariants. Assuming the network has been properly updated, everything continues to operate with the new failures and the updated policy. If the latest policy violates an invariant, the information is logged for the network operator to analyze, the failure is undone, and the previous policy can be reinstalled in the network.

3. PROOF OF CONCEPT

Our proof of concept consists of creating a scenario in which the controller incorrectly installs a forwarding loop in the network upon the failure of a link.

The figure depicts a route in red from H1 to H2 that is configured by the SDN Controller, while the backup route, in green, is the path that the network should take upon the injection of failure; the backup route is pushed when the S4-S5 link is severed. When OVX reports the updated flows to NetPlumber, there are two outcomes depending on the controller and policies in question. A transient forwarding loop can be created, if the controller pushes flows to the switches in the order: S1, S6, S4, S2, S5. In this case, since the original red route flows have not yet been removed by the controller, there will be a transient forwarding loop circulating between S2, S3, and S4. On the other hand, no network invariants would be violated if the routes are pushed in the order: S5, S2, S4, S6, S1. The Chaos Monkey algorithmically decides on a link to fail and detects the violated network invariant.

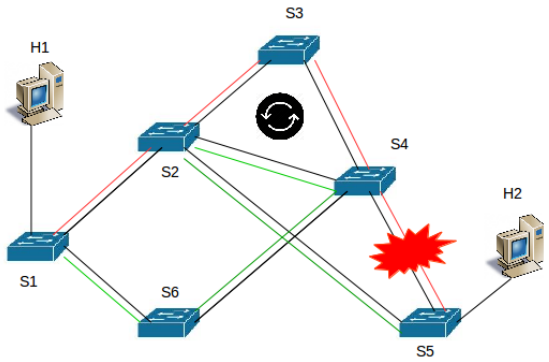

Figure 2: Example topology with a forwarding loop.

4. CONCLUSION

In this paper, we have laid the foundation for post-deployment failure injector for SDN Controllers. While this testing paradigm has been used in various cloud computing applications, this is the first step towards developing a solution for realtime SDN Controller verification. With many classes of network failure, it will be a continued effort to extend the SDN Chaos Monkey such that we can demonstrate coverage across possible failure classes. Developers of the controller platform will be able to use the SDN Chaos Monkey tool to test how the controller responds to any sequence of failures that could reasonably occur in the network. There are truly an endless amount of failures that could appear in the network, and it would be unreasonable for anyone – even Chaos Monkey – to cover them all. However, the Chaos Monkey utility encourages a testing paradigm that leads to continual and constant improvement on a complex piece of software upon which the entire network depends on. As the SDN controller becomes a more reliable and stable software agent, the SDN movement will gain further credibility and continue to disrupt the landscape of networking.

5. REFERENCES

[1] M. Canini, D. Venzano, P. Peresini, D. Kostic, J. Rexford *et al.*, "A nice way to test openflow applications." in *NSDI*, vol. 12, 2012, pp. 127–140.

[2] C. Bennett and A. Tseitlin, "Netflix: Chaos monkey released into the wild. netflix tech blog," 2012.

[3] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "Openvirtex: A network hypervisor," *Open Networking Summit*, 2014.

[4] P. Kazemian, M. Chan, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis." in *NSDI*, 2013, pp. 99–111.