ECE 542 Final Project
Creating an SDN Fault Injector
Ben Ujcich and Uttam Thakore
SDN Definition and Architecture [1]

1) Forwarding decisions decoupled
2) Centralized control
3) Programmable network
Static and Dynamic SDNs

- **Static**
  - Flow rules determined ahead of time
  - Flows instantiated and mostly left alone
  - Network topology relatively stable
  - Use cases: power grid communications network

- **Dynamic**
  - Flow rules determined at run time
  - Flows instantiated and removed as needed
  - Network topology more subject to change
  - Use cases: university campus enterprise network

However, most networks have characteristics of both, so ‘static’ and ‘dynamic’ are really a spectrum
Selected OpenFlow protocol packets [2]

- **Asynchronous (switch to controller)**
  - `PACKET_IN`: packet doesn’t match any flow in flow table
  - `FLOW_REMOVED`: flow expired due to idle or hard timeout

- **Controller-to-switch**
  - `FLOW_MOD`: add a new flow entry
  - `PACKET_OUT`: send packet back to the switch
  - `STATS_REQUEST`: request switch statistics
  - `STATS_REPLY`: receive switch statistics

- **Symmetric**
  - `HELLO`: initial handshaking between switch and controller
  - `ECHO_REQUEST`: used for liveliness
  - `ECHO_REPLY`: used for liveliness
Motivation for SDN Fault Injection

- **Static SDN networks**
  - Ensure that *initial network flow rules* are set up correctly before handing off the network control plane

- **Dynamic SDN networks**
  - Ensure that *flow table misses and newly instantiated flows* are set up correctly and reflect the intended state of the network

- **State consistency**: is the state of the switches the same as the state of the controller?

- **Fault propagation**: how far do faults manifest themselves when they propagate to other SDN planes?

- **Security**: how easily can the SDN be hacked?
Prior Work in SDN Fault Injection

- “Advanced Study of SDN/OpenFlow controllers” [3]
  - See how various SDN controllers responded to malformed OpenFlow headers (incorrect message length, invalid OpenFlow version, incorrect OpenFlow message type) and OpenFlow messages (\texttt{PACKET\_IN} and port status)
  - Not representative of the most common messages or scenarios
  - No study of how the fault propagates to other SDN planes (e.g. to the network application)

  - Proposes various potential security vulnerabilities
  - Does not implement them in experimentation
Project Idea

Create a fault injector for SDN control plane

1. Investigate the OpenFlow v1.0 control plane protocol
   - Types of packets defined in the OpenFlow standard
   - Header structure of OpenFlow protocol packets

2. Implement fault injector within an SDN network
   - Design and configure a network topology that allows for fault injection
   - Use resources on the NSF’s GENI networking testbed
   - Use existing tools in a novel way to implement a fault injector specifically targeted for SDN control plane traffic
   - See how faults propagate up through SDN planes
Problem: Where is the most ideal place to position the fault injector? On the controller? On the switch?
Solution: Design an inline, man-in-the-middle fault injector that intercepts and modifies OpenFlow protocol packets.
Solution benefits: Controller and switch agnostic, inline capabilities
Limitations of Scope

- No fault injection in the northbound API interface (network application – controller interface)
  - Northbound API is not standardized among controllers
  - Southbound API is well-defined, standardized (as OpenFlow protocol), and widely used

- No fault injection within the SDN controller
  - Dependent upon language (e.g. Floodlight in Java, POX in Python, NOX in C++)

- No fault injection within the SDN switches
  - Highly dependent on implementation (software or hardware)
  - Some implementation details are vendor proprietary
Implementation

- Global Environment for Network Innovations (GENI) networking testbed
  - Rapid prototyping of network topologies
  - More flexible than network simulators (e.g. Mininet)

- Fault injector written in Python

SDN Controller node
  Floodlight

Fault Injector node

SDN Switch node
  Open vSwitch

Host node

Host node
Project Architecture

Net App 1 → SDN Controller → Fault Injector → SDN Switch → Host 1
Net App 2
Net App n

APPLICATION PLANE

CONTROL PLANE

DATA PLANE

OpenFlow protocol for controller-switch communications
Implementation Tools

- **Floodlight** – Java-based OpenFlow controller with REST NB API
- **Open vSwitch** – software-based OpenFlow switch
- **Scapy** – packet manipulation library for Python to manipulate OpenFlow protocol packets on fault injector node
- **brctl** – Ethernet bridge administration for Linux to enable bridging on fault injector node
- **iptables** – firewall and gateway administration for Linux to enable forwarding on fault injector node
- **nfqueue-bindings** – iptables userspace interface for interacting with packets in Python to only capture OpenFlow protocol packets on fault injector node
- **POX** – Python-based OpenFlow controller to enable OpenFlow packet parsing within the fault injector
Experiment

- Intercepting **STATS_REPLY** OpenFlow packets to check controller’s view of the network state
- Used in learning switch forwarding setup:
  1. Host A pings Host B.
  2. Switch does not have any associated flow entries between hosts A and B. Switch sends packet to controller via **PACKET_IN** message.
  3. Controller sends a **FLOW_MOD** message. Switch receives it.
  4. Network application queries controller for list of flows. Controller sends a **STATS_REQUEST** message to switch.
  5. Switch receives **STATS_REQUEST** message and returns a **STATS_REPLY** message with the newly instantiated flow list.
  6. Fault injector intercepts **STATS_REQUEST** and removes flow list.
  7. Controller thinks network state is different than it actually is.
Experimental Results

- **Case 1:** \texttt{STATS\_REPLY} always injected, beginning at controller startup
  - Controller starts handshake with switch but cannot receive reply
  - \textbf{Controller thinks that no switches exist}

- **Case 2:** \texttt{STATS\_REPLY} selectively injected, beginning after controller startup
  - Network application queries the controller for current flows
  - \texttt{STATS\_REQUEST} messages to switches
  - Northbound API returns empty flow list based on empty \texttt{STATS\_REPLY} from the southbound API
  - Flow entries previously instantiated still show up as evidenced by looking at Open vSwitch flow table.
  - \textbf{State inconsistency and fault propagation between switch and controller}
Insights and Future Work

- Fault tolerance in the SDN control plane has wide-ranging implications for control plane security
  - Man-in-the-middle attack possible on unencrypted control plane traffic [4]

- How else can the fault injector tool be used?
  - Monitoring: one of many data sources to compare SDN network state between the fault injector, switches, and controller
  - Auditing: keep a record of all control plane actions for state inspection
  - Response: block suspicious control traffic
Questions?
References


