

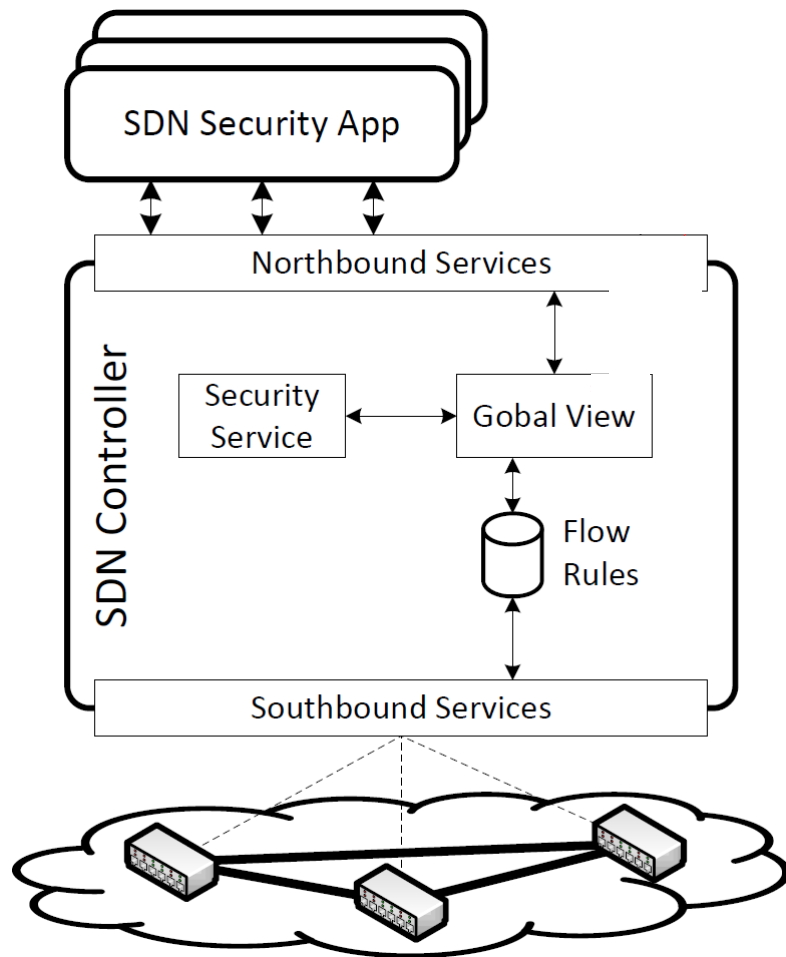
Preventing Malicious SDN Applications From Hiding Adverse Network Manipulations

Christian Röpke and Thorsten Holz
Ruhr-University Bochum

Problem Statement

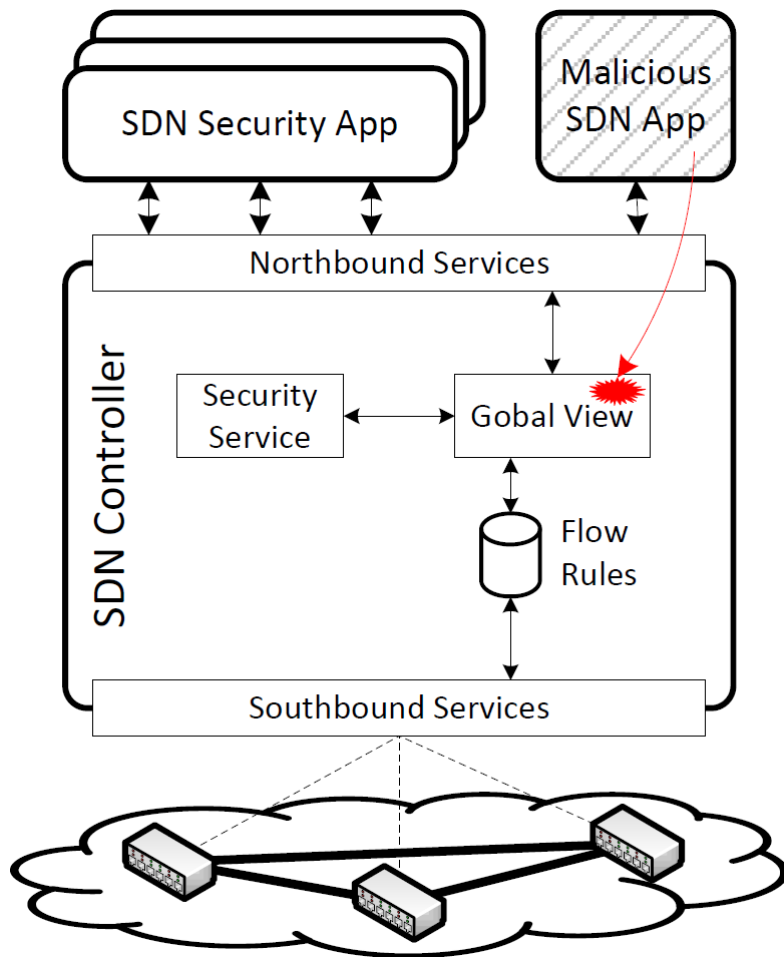
- Malicious SDN applications are capable of intensively manipulating internal data structures of SDN controllers.
- As a result, attackers can hide malicious actions, e.g., from SDN controller security services and SDN security applications.

Hiding Adverse Network Manipulations



- SDN apps insert/remove rules.
- SDN security apps and SDN controller security services keep track on manipulations by observing the global view.
- Normal case:
 - A security mechanism raises an alert in case an observed manipulation violates a policy.

Hiding Adverse Network Manipulations



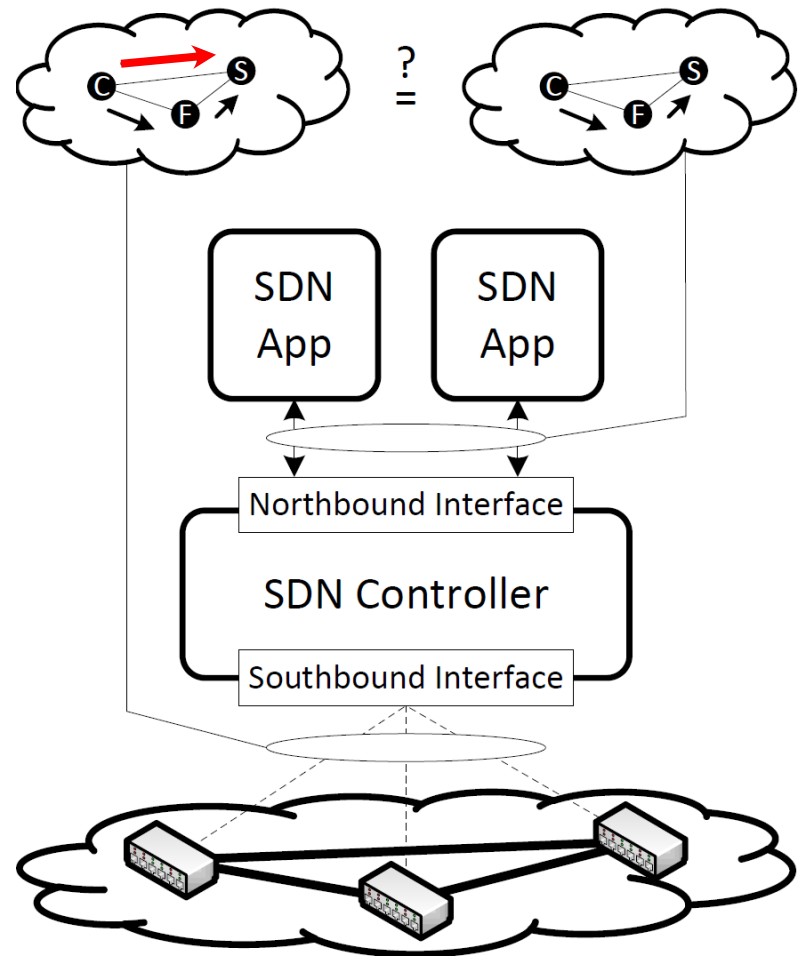
- SDN apps insert/remove rules.
- SDN security apps and SDN controller security services keep track on manipulations by observing the global view.
- Normal case:
 - A security mechanism raises an alert in case an observed manipulation violates a policy.
- Compromised SDN controller:
 - Global view is tampered to hide adverse manipulations.
 - **A security mechanism is unable to notice adverse network changes anymore!**

Current Solutions

- SDN application sandboxing
 - Put SDN applications into sandboxes and restrict access to critical operations to prevent SDN controller attacks
 - Problems:
 - Adversely usable critical operations must be known beforehand
 - Distinguishing between benign and malicious use is rather difficult
- Policy checking
 - Intercept network programming attempts and drop the ones which violate a security policy
 - Problems:
 - Defining security policies in complex networks is error-prone
 - A single missing policy can result in an attack

Our Approach

- Intercept network programming attempts
- Before programming a switch compare the actual network state with the one provided by a potentially compromised SDN controller
- Prevent actively hidden malicious flow rule insertions and removals
- No need for
 - Complex analyses
 - Error-prone configurations



Assumption

- Malicious SDN applications hide adverse network manipulations as soon as possible and as long as possible.
- This is reasonable because
 - A security mechanism can easily observe malicious actions and raise alerts, and
 - Attackers have an immense interest in remaining undetected.

Challenges

- Take southbound protocol specifics into account (i.e., barrier requests)
 - Example:
 - Add flow rule 1 -> Barrier request -> Add flow rule 2
(Goal: install flow rule 1 before flow rule 2)
 - Possible race condition:
 - If a barrier request is not handled appropriately, flow rule 2 is installed at first.
 - This is the case, e.g., when checking flow rule 1 takes longer than checking flow rule 2.

Challenges

- Take southbound protocol specifics into account (i.e., barrier requests)
- Get global network view from SDN controller (e.g., via REST API)
 - Efficiently
 - To keep the network programming delay at a minimum

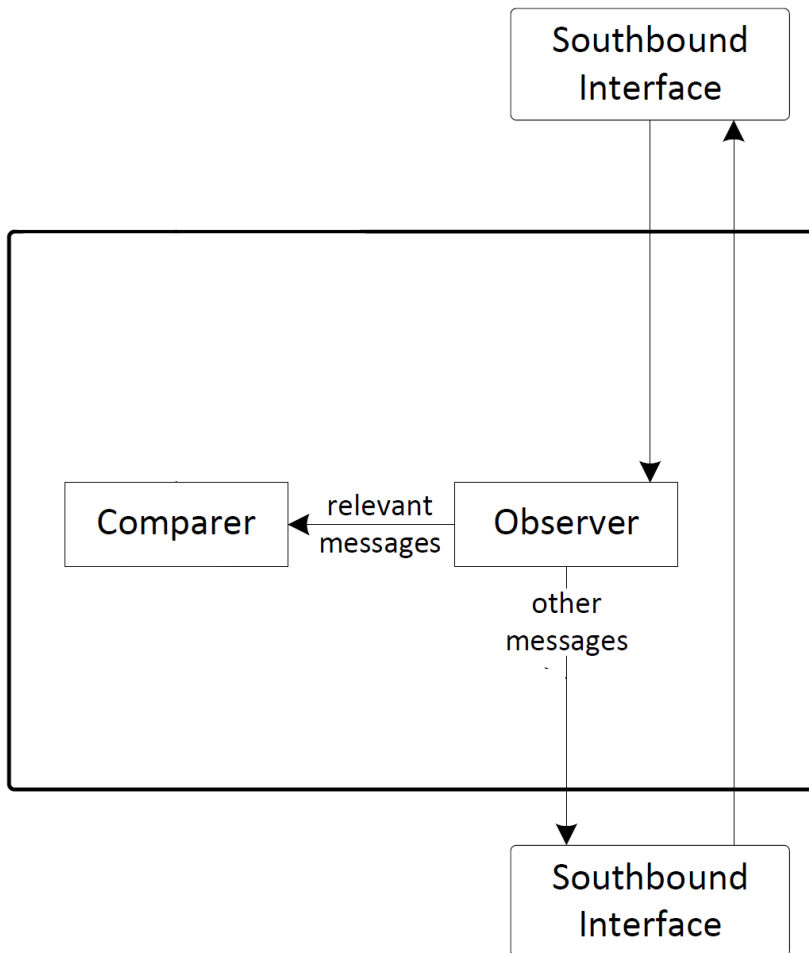
Challenges

- Take southbound protocol specifics into account (i.e., barrier requests)
- Get global network view from SDN controller (e.g., via REST API)
 - Efficiently
 - Indistinguishably from other SDN applications
 - Otherwise, corresponding requests can be handled differently by an attacker, which may result in a bypass

Challenges

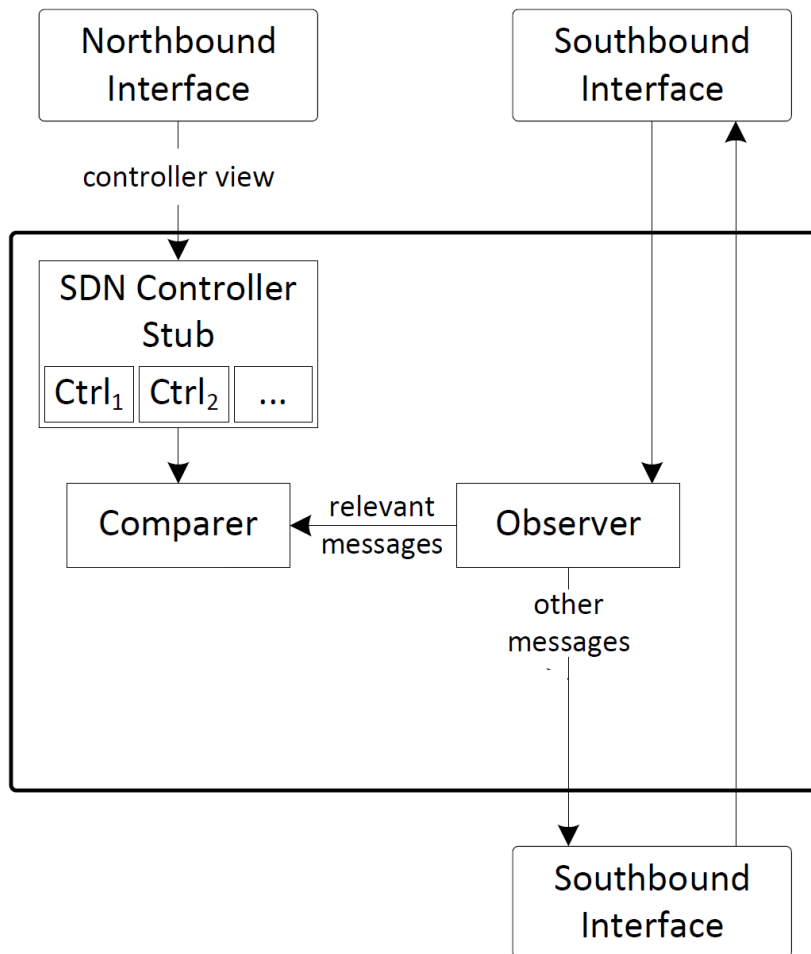
- Take southbound protocol specifics into account (i.e., barrier requests)
- Get global network view from SDN controller (e.g., via REST API)
 - Efficiently
 - Indistinguishably from other SDN applications
- Compare network states efficiently
 - To keep the network programming delay minimal

Architecture



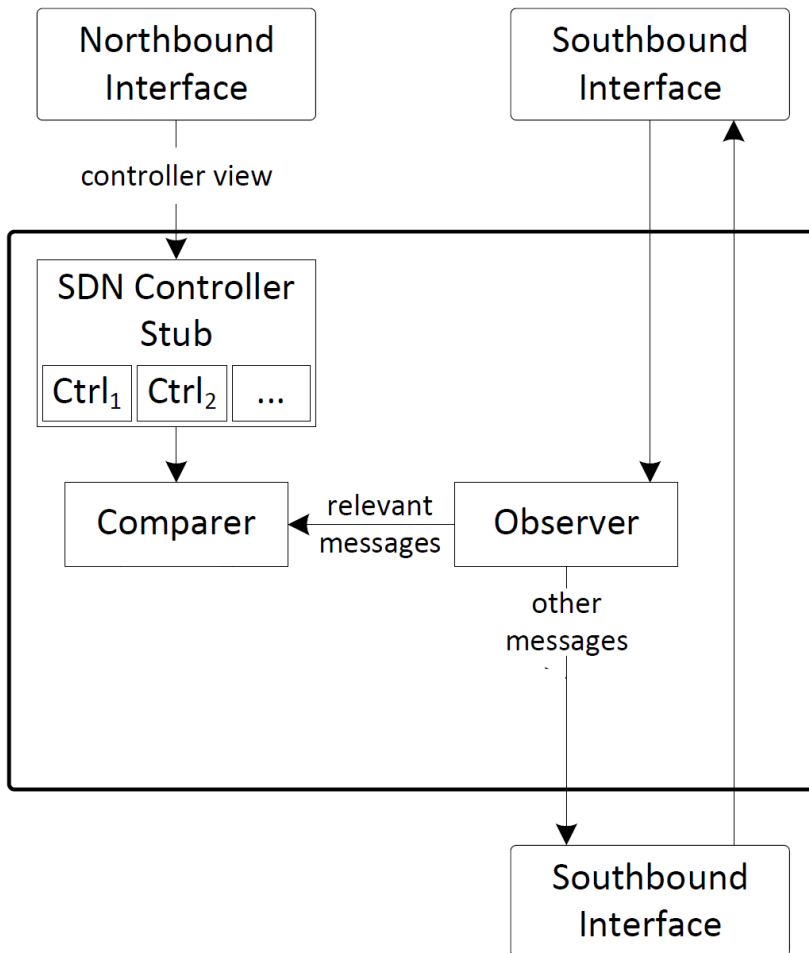
1. Observer intercepts relevant messages and passes them to a comparer unit.

Architecture



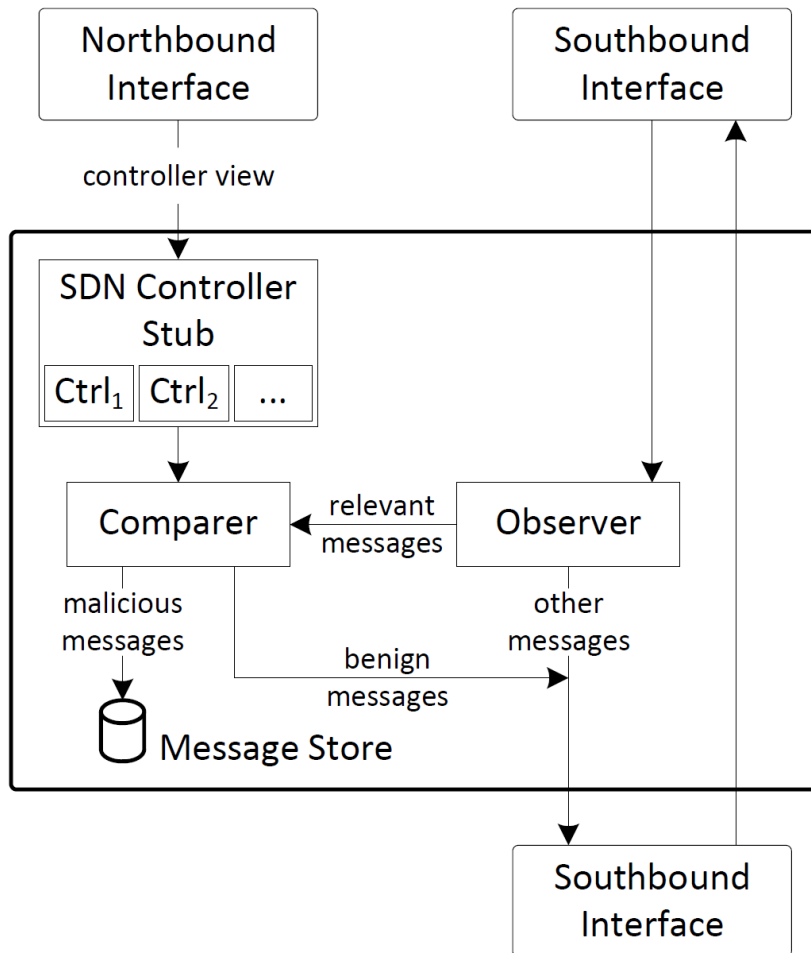
1. Observer intercepts relevant messages and passes them to a comparer unit.
2. Comparer queries global network view via an SDN controller stub.

Architecture



1. Observer intercepts relevant messages and passes them to a comparer unit.
2. Comparer queries global network view via an SDN controller stub.
3. Comparer transforms inputs to a format which enables efficient network state comparison.
4. Comparer checks if the network change in question is made visible by the SDN controller.

Architecture



1. Observer intercepts relevant messages and passes them to a comparer unit.
2. Comparer queries global network view via an SDN controller stub.
3. Comparer transforms inputs to a format which enables efficient network state comparison.
4. Comparer checks if the network change in question is made visible by the SDN controller.
5. If yes, change is considered benign and thus forwarded. If no, change is considered malicious, dropped, and stored for further investigations.

Implementation

Challenge

- Take southbound protocol specifics into account

Proposed Solution

- Block control channel in case of barrier requests
 - Ensure message ordering as instructed by SDN controller
 - Delay switch programming for the checking time

Implementation

Challenge

- ✓ Take southbound protocol specifics into account
- Appear as normal SDN application

Proposed Solution

- Use REST API
- Typical feature of SDN controllers
- All kinds of SDN applications can use this interface w/o disclosing their purpose

Example:

```
curl -u onos:rocks http://<ip>:8181/onos/v1/flows/of:<dpid>/<cookie>  
curl -u onos:rocks http://<ip>:8181/onos/v1/flows/of:<dpid>/<cookie>
```

Implementation

Challenge

- ✓ Take southbound protocol specifics into account
- ✓ Appear as normal SDN application
- Get global network view efficiently

Proposed Solution

- Ask SDN controller only for a relevant subset of installed flow rules
 - This is possible, e.g., by using the cookie value which can identify a flow-mod message
 - If supported by an NBI, an SDN application can query a flow rule by its cookie value

Implementation

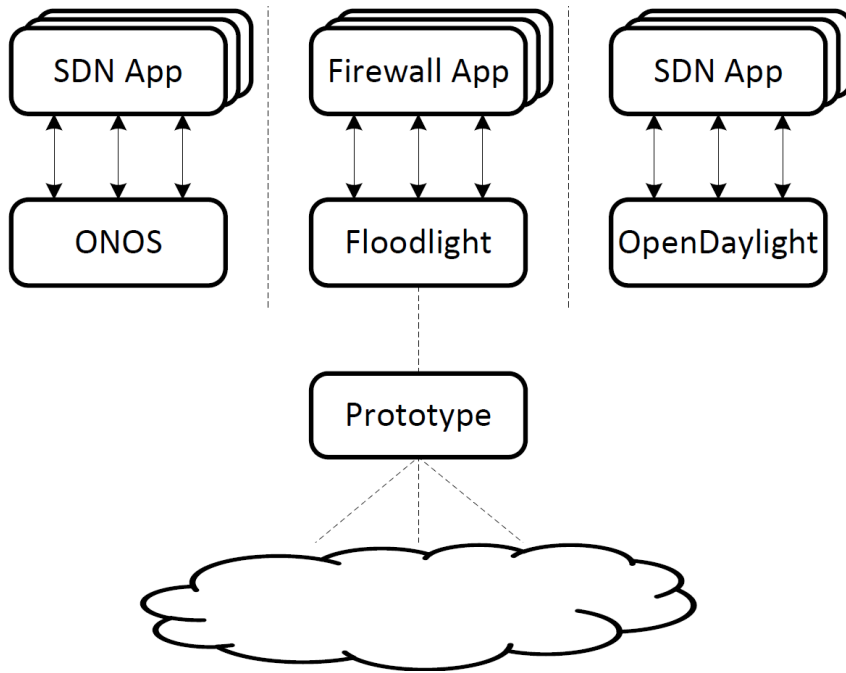
Challenge

- ✓ Take southbound protocol specifics into account
- ✓ Appear as normal SDN application
- ✓ Get global network view efficiently
- ✓ Compare network states efficiently

Proposed Solution

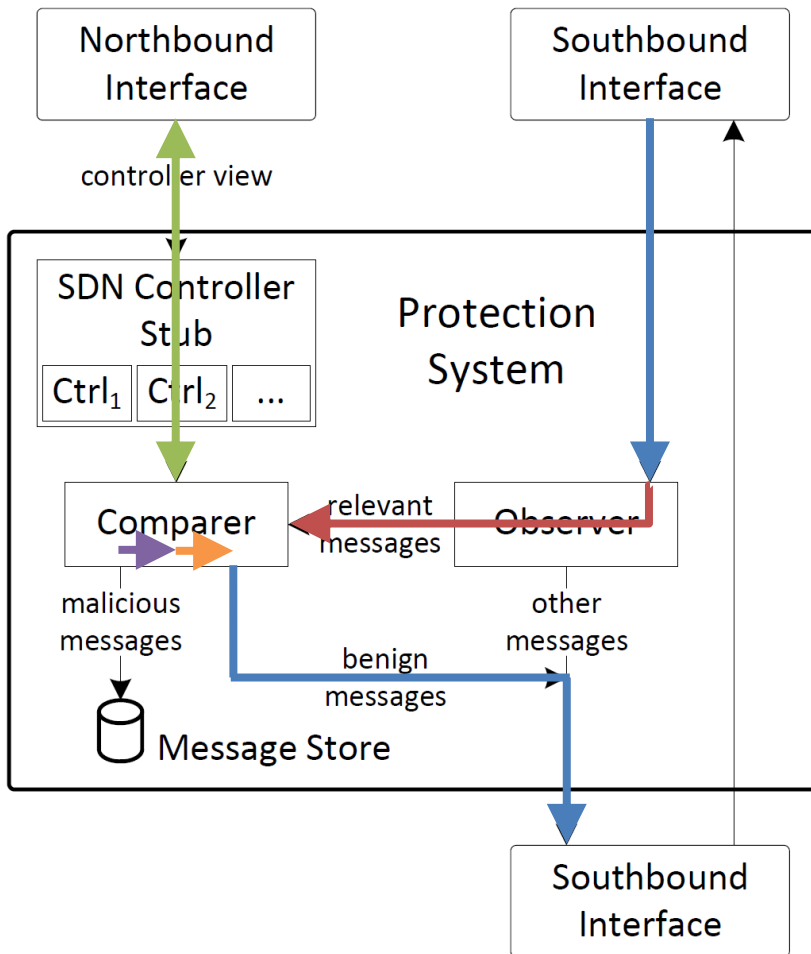
- Generate a hash value for flow rule in question and flow rule(s) provided by the SDN controller
- Hash value comparison is typically fast

Evaluation



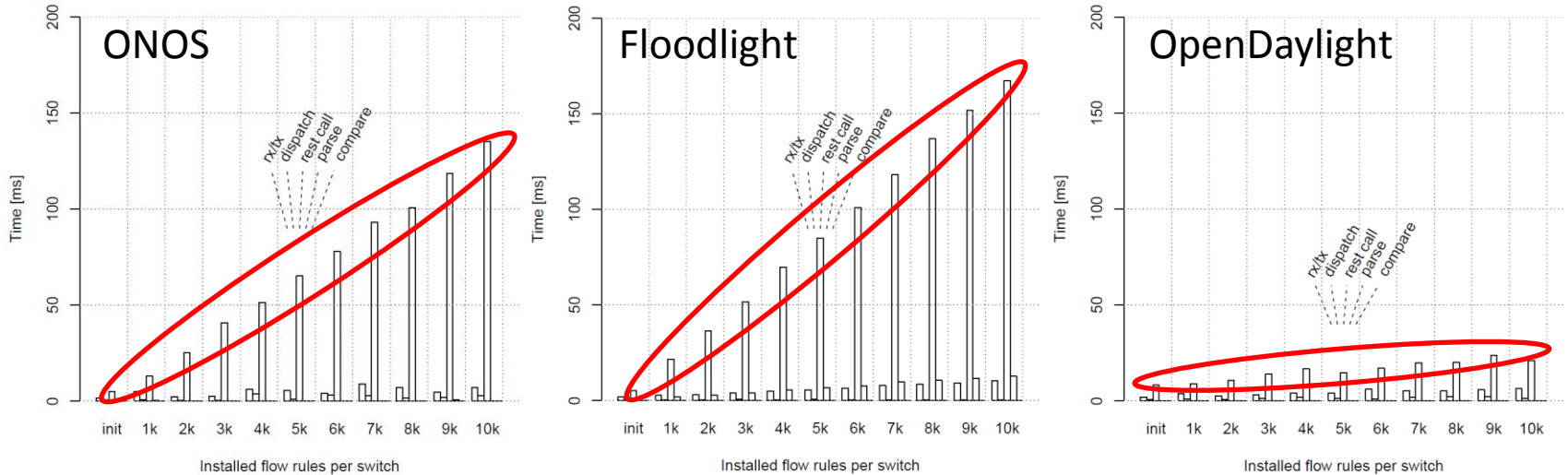
- ONOS, Floodlight and ODL as SDN controllers
- Various network topologies are simulated via Mininet
- Network load is simulated via increasing the number of added rules per switch
- Our prototype sits between SDN controller and network
- Test system: 2x Intel Core i7, 32 GB RAM, Ubuntu 16.04

Evaluation



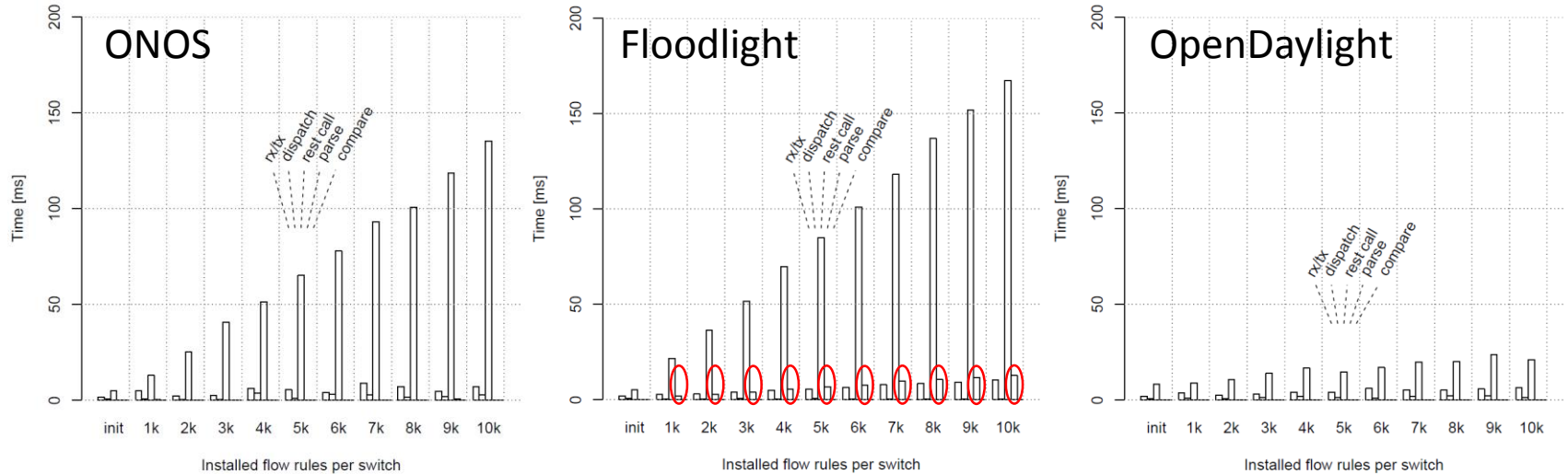
- Important factors:
 - $t_{rx/tx}$: time to receive / send traffic from SDN controller / to network device
 - $t_{dispatch}$: time to dispatch programming attempts to a comparer thread
 - $t_{rest\ call}$: time to query the global network view
 - t_{parse} : time to parse global network view and to generate hashes
 - $t_{compare}$: time to compare network states

Results



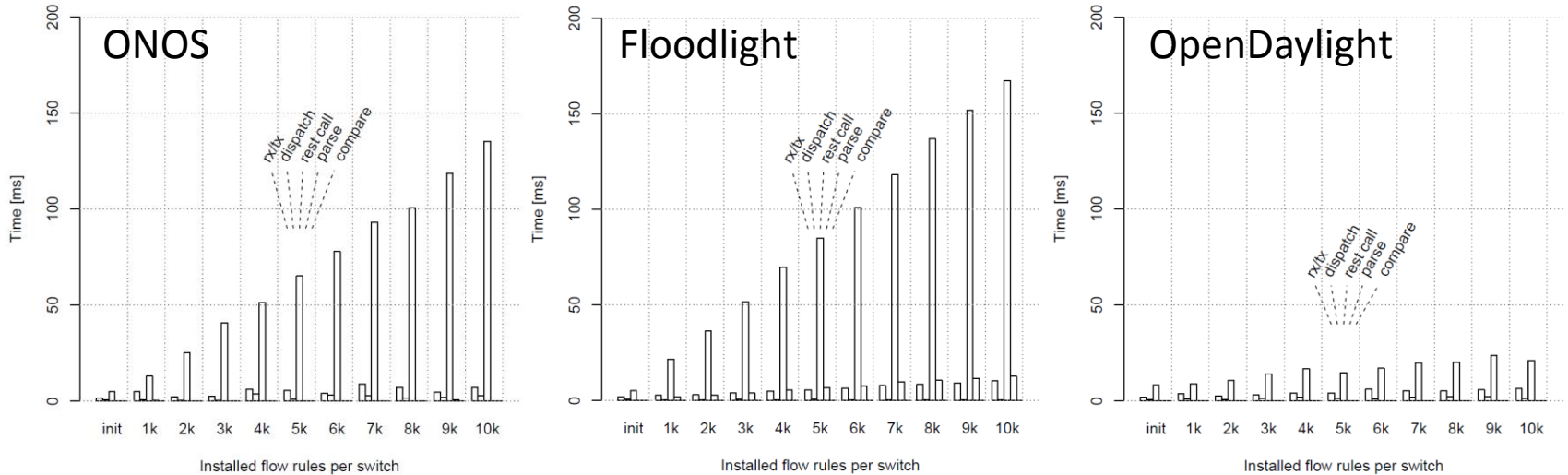
- By far the slowest part is to get the global network view via REST calls
 - Delay grows slowly (OpenDaylight) to moderately (ONOS, Floodlight)
 - OpenDaylight's responds are the fastest
 - Extend NBI by a mechanism faster than REST

Results



- By far the slowest part is to get the global network view via REST calls
 - Delay grows slowly (OpenDaylight) to moderately (ONOS, Floodlight)
 - OpenDaylight's responds are the fastest
 - Extend NBI by a mechanism faster than REST
- Parsing the global network view is fast except for Floodlight
 - Its responds contain all installed flow rules
 - Extend Floodlight's NBI to query individual flow rules

Results



- By far the slowest part is to get the global network view via REST calls
 - Delay grows slowly (OpenDaylight) to moderately (ONOS, Floodlight)
 - OpenDaylight's responds are the fastest
 - Extend NBI by a mechanism faster than REST
- Parsing the global network view is fast except for Floodlight
 - Its responds contain all installed flow rules
 - Extend Floodlight's NBI to query individual flow rules
- Time to compare network states is negligible

Limitation

- Our prototype provides full support for ONOS, but only part support for Floodlight and ODL.
- Reason:
 - Adding/removing flow rules can also be achieved without using an internal flow rule database.
 - Querying corresponding flow rules requires additional control traffic, which is blocked in case of barrier requests.

Conclusions

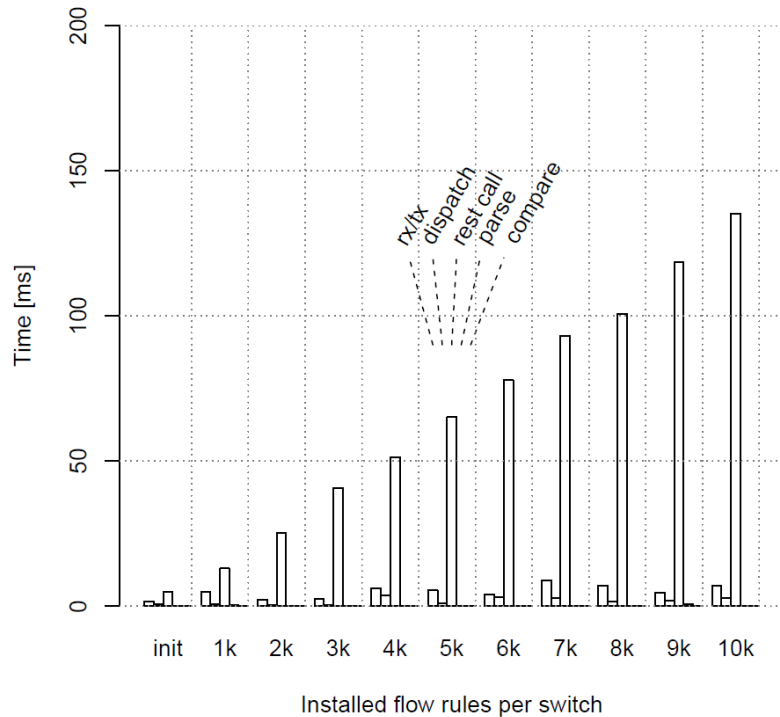
- Malicious SDN applications are capable of hiding adverse network manipulations.
- We complement existing security solutions by revealing and preventing such attacks.
- Our performance evaluation indicates efficiency which allows proactive security.



Thank you!

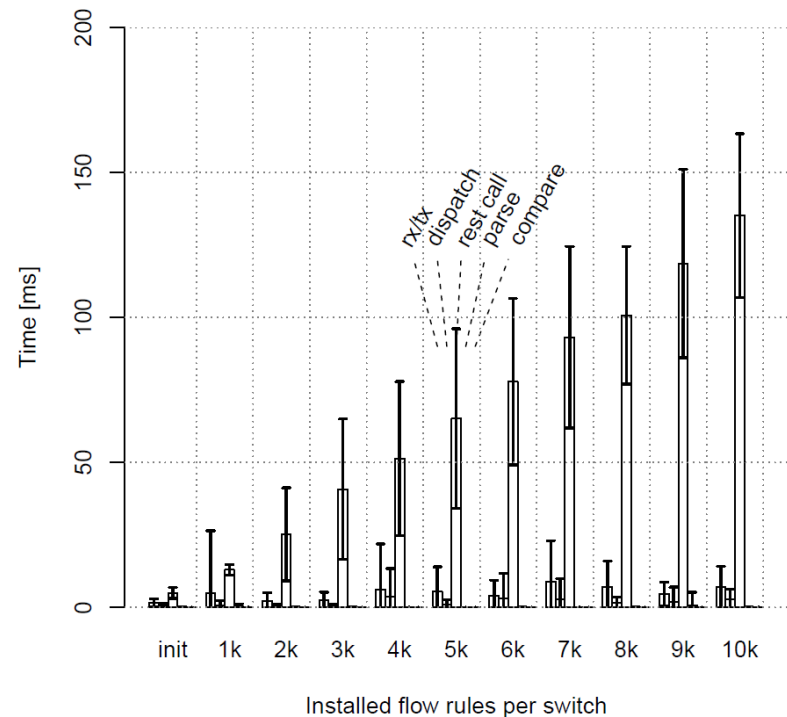
Backup

Average delay for adding flow rules with ONOS



Average delay and standard deviation for ONOS

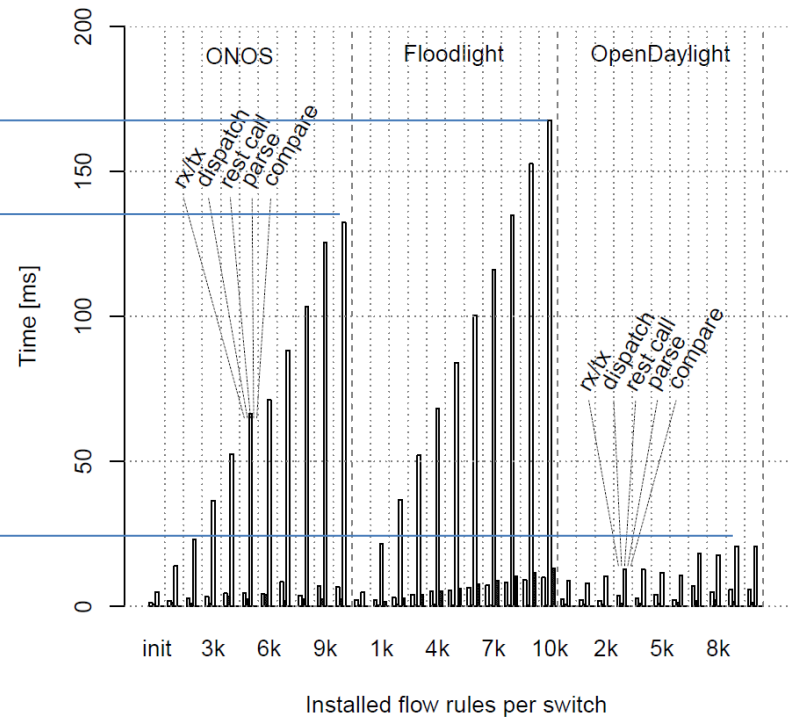
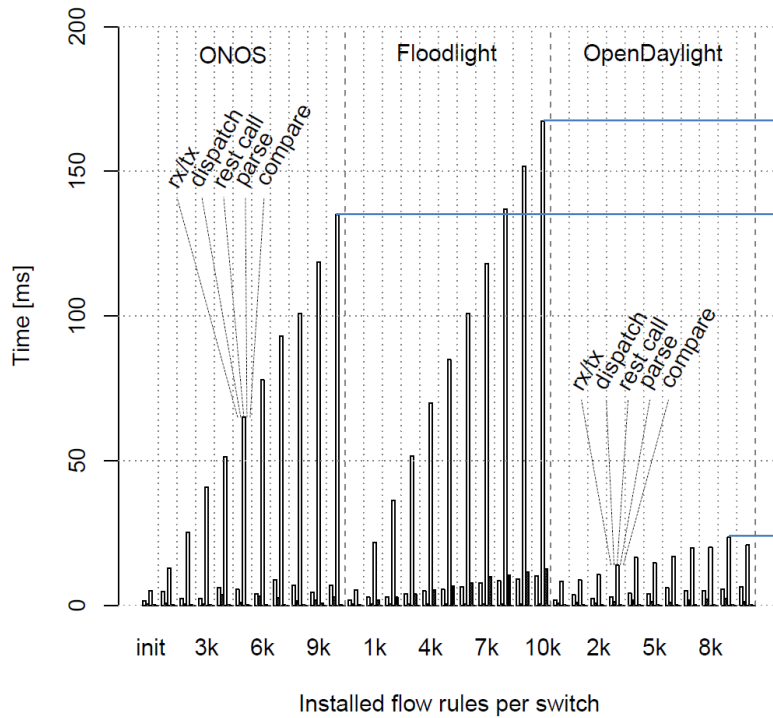
225ms -> X



Backup

Adding Flow Rules

Removing Flow Rules



Backup

- ONOS
 - Adding a flow rule
 - A db entry is created with status *pending_add*
 - Then, a flow-mod and barrier-request is sent
 - After receiving barrier-reply, this db entry's status changes to *added*
 - When intercepting flow-mod's, corresponding flow rules are supposed to be visible in ONOS's global network view
 - If this is not the case, then they must be actively hidden
 - Removing a flow rule
 - A db entry's status changes to *pending_remove*
 - Then, a flow-mod and barrier-request is sent
 - After receiving barrier-reply, this db entry is removed
 - When intercepting flow-mod's, corresponding flow rules are supposed to be visible with status *pending_remove*
 - If this is not the case (*added*), then their removal is actively hidden

Backup

- No full support for Floodlight and ODL
- Reason:
 - Adding/removing flow rules can also be achieved without using an internal flow rule database.
 - Querying such flow rules requires additional control traffic which we blocked in case of barrier requests.
- Possible solution:
 - In case the main connection is blocked, establish an auxiliary connection to exchange flow-stats messages.
 - Inject/remove data about the flow rule under validation in flow-stats replies to complete the SDN controller's global view.
 - In case of a benign network change, injected/removed data is correct.
 - In case of a malicious change, injected/removed data is hidden.