SDNShield: Reconciliating Configurable Application Permissions for SDN App Markets

Yan Chen

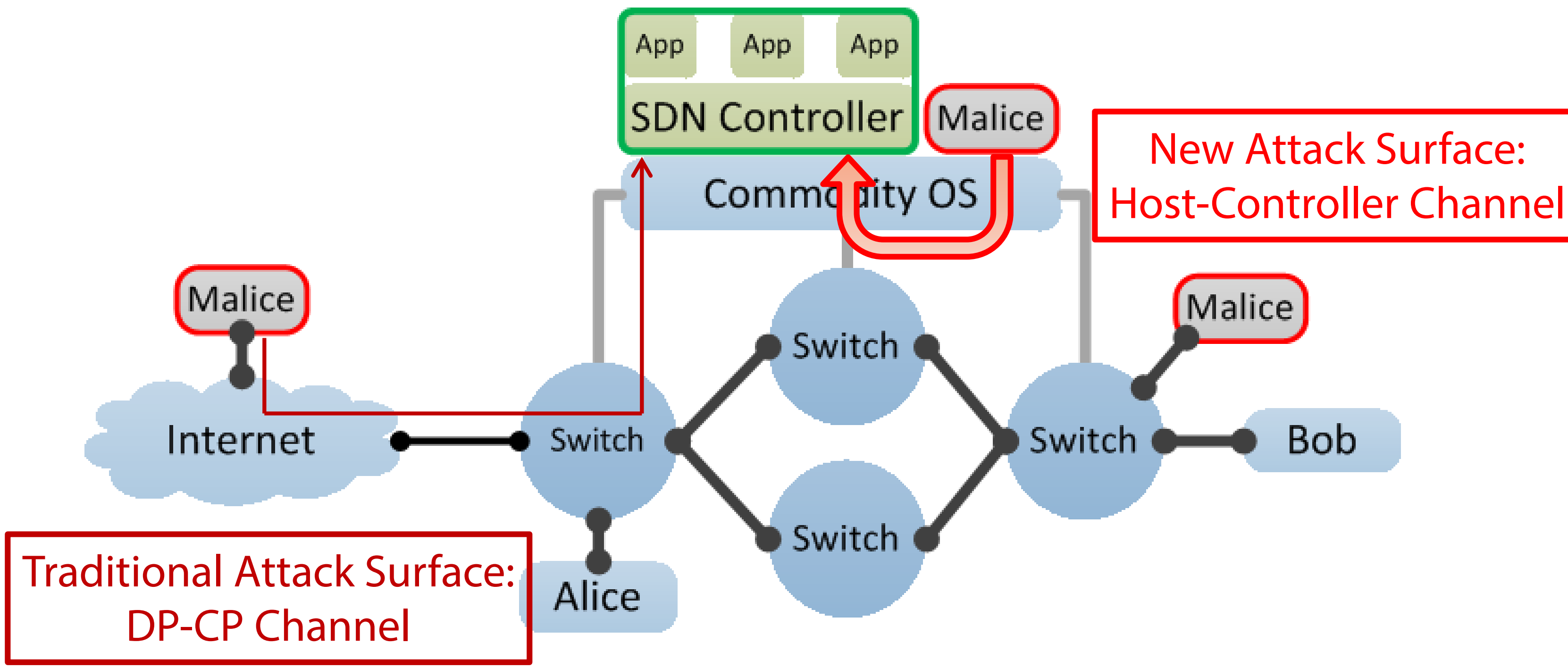
Lab of Internet and Security Technology (LIST)

Northwestern University, USA

Joint work with Xitao Wen, Bo Yang Chengchen Hu, Yi Wang, Bin Liu, and Xiaolin Chen

Motivation

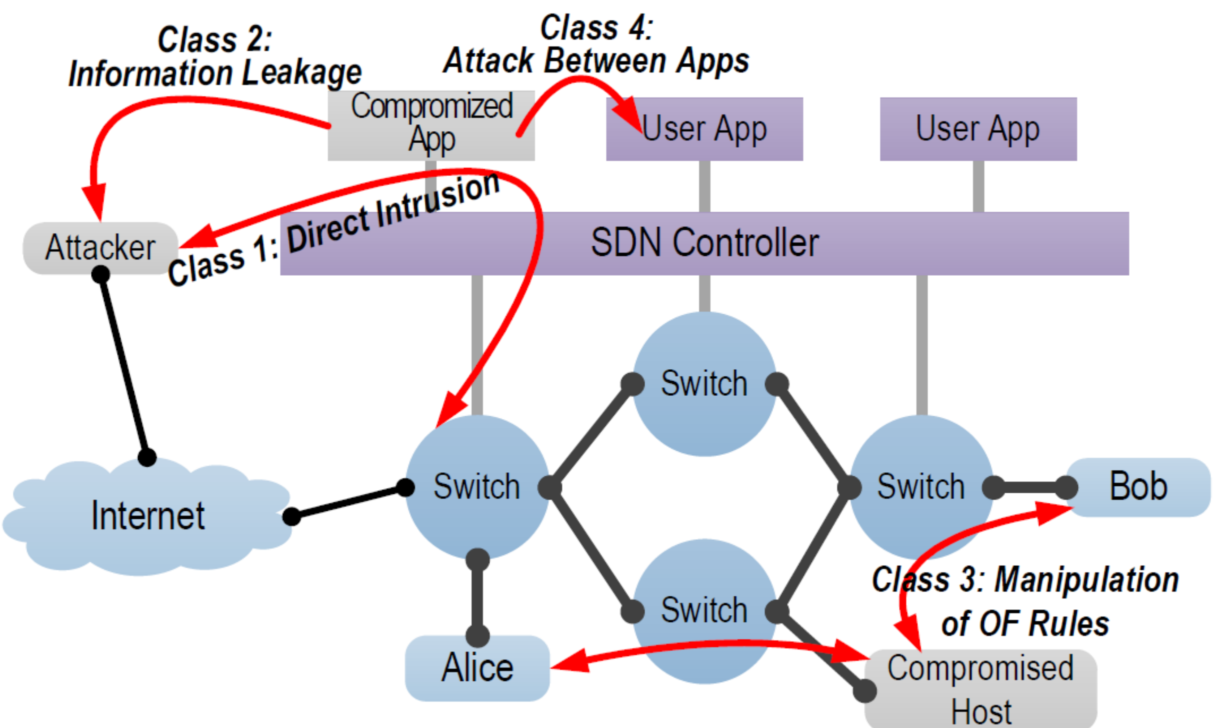
- SDN security concerns rank No.1 road blocker for SDN adoption*
- Over-privilege problem in control plane



* *http://searchsdn.techtarget.com/feature/Five-reasons-IT-pros-are-not-ready-for-SDN-investment*

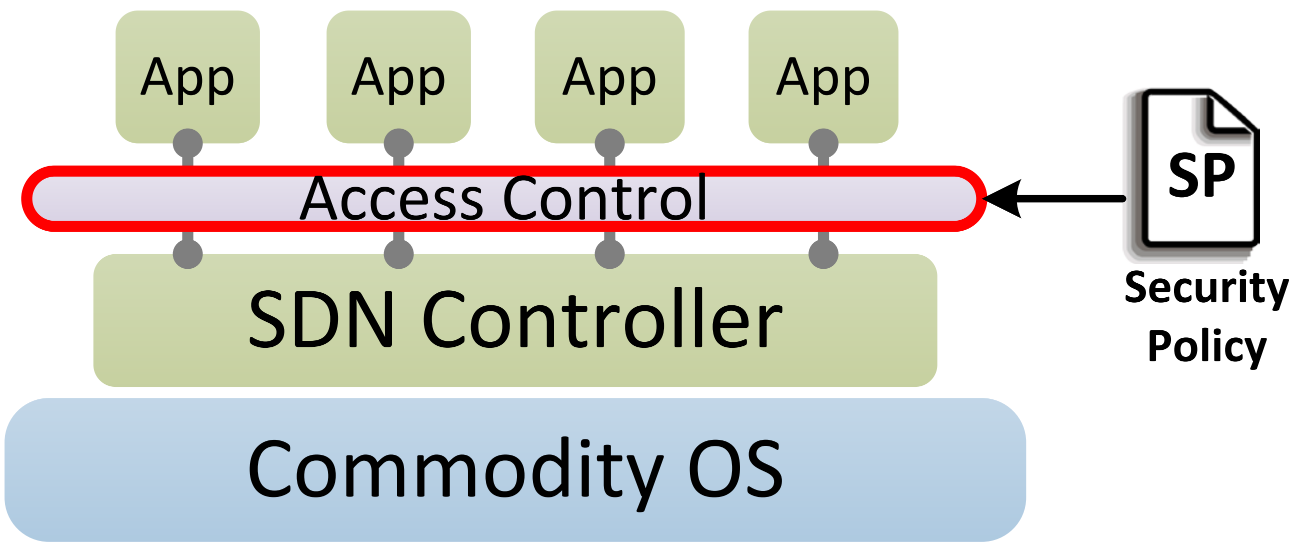
Threat Models

- Exploit of existing benign-but-buggy apps
- Distribution of malicious apps by attacker
- Plenty of potential attacks



Approach

- Policy-based Access Control on Apps
  - Proactively eliminate apps' over-privilege behaviors



Existing SDN Security Systems

- Cryptographic authentication
  - Mainstream controller platforms
  - No isolation
- Android-like permissions
  - SE-floodlight[NDSS15], FortNOX[HotSDN12]
  - Too coarse grain
- Strong & heavy isolation: Rosemary [CCS14]

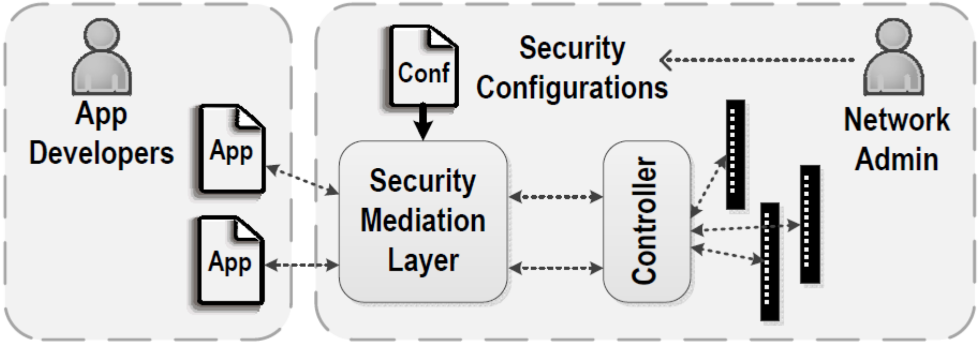


Figure 4.1. Existing SDN access control systems.

- Open Question: how to have the developer and network admins collaborate to enforce security policies on apps ?

Our Vision

- Flexible permission abstractions
  - App developers can express fine-grained permission requests.
- Limited increase on management burden
  - Administrators can easily refine app permissions with higher-level security policies.
- Reliable and lightweight enforcement
  - The controller needs a secure while efficient isolation architecture to enforce permissions.

Challenge

1. How to describe SDN app based permissions?
   - Accurately describe the complex API behavior space
   - Complicated logic is needed to depict inter-dimensional relations

SDNShield Permission Abstractions

2. How to reduce burden of admin on drafting security policy?
   - Need to reconcile inputs from app developer and network admin
   - Need a bridge to reshape app's permission space with local security requirements

SDNShield Security Policy Reconciliation

3. How to reliably enforce permissions?
   - Runtime isolation
   - Reference monitoring

SDNShield Isolation Architecture

System Overview

- Permission Manifest
  - Describes per-app permission requirement
  - Written in permission language
  - Drafted by app developer
  - Reviewed by controller vendor
- Security Policy
  - Describes security requirements of local environment
  - Written in security policy language
  - Provided by network admin

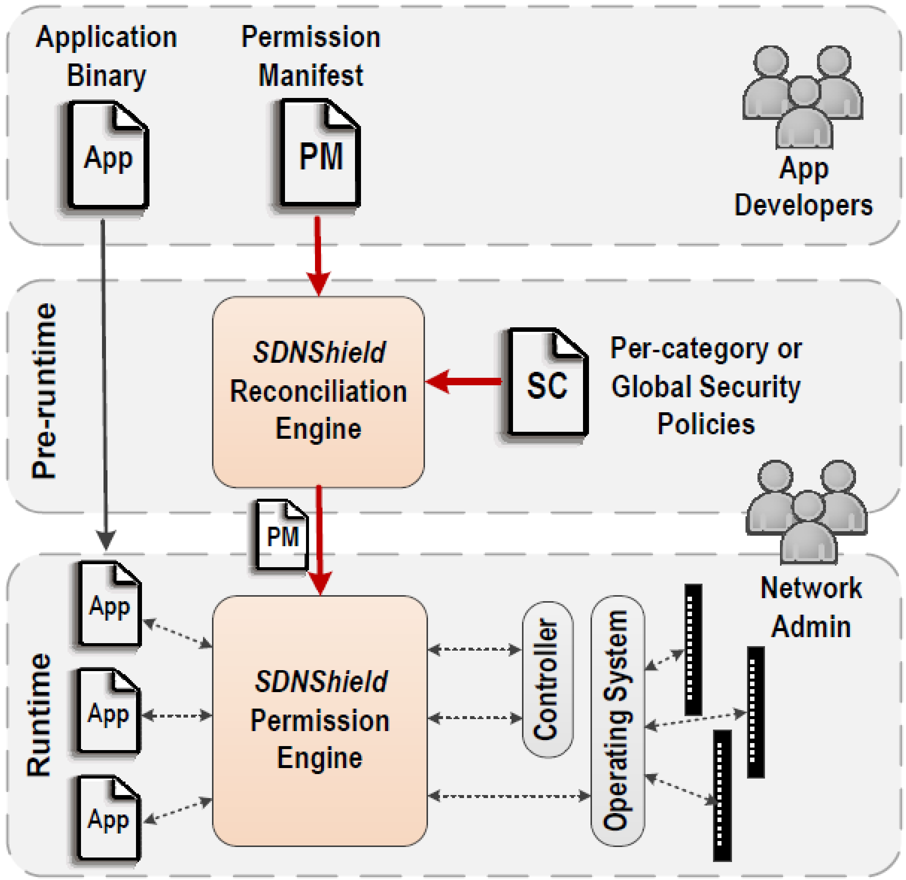


Figure 4.2. *SDNShield* overview.

Roadmap

- SDNShield System Design
- SDNShield Implementation and Evaluations

Published in HotSDN2013 and DSN2016

- Ongoing Extension to REST APIs and NFVs

- Conclusions

Permission Abstractions

Permission Token

```
PERM insert_flow LIMITING \
WILDCARD IP_DST 0.0.0.255 AND \
( FORWARD OR DROP )
```

Permission Filter

Permission Abstractions

- Coarse-grained permission tokens
  - Describe chunks of logical resources
  - E.g., read_flow_table, insert_flow, visible_topology, send_pkt_out, etc.
- Fine-grained permission filters
  - Predicates on permission tokens connected with logic operators
  - *Flow filters*: flow match, flow action, flow wildcard, flow ownership
  - *Topology filters*: partitioning physical topology, virtual topology
  - Priority filter, statistics filter, event filter, etc.

Security Policy Reconciliation

- Goal: reduce administration burden on permission review
- Security Policy Language
  - Based on concepts that administrator are already familiar with
  - Three major abstractions:
    - Permission boundary
    - Mutual exclusion
    - Permission Customization
- Algorithm-assisted reconciliation process
  - Admin feeds customization conditions and local security policy
  - Reconciliation engine finds policy violation and provides permission candidates

Security Policy Examples

- Example 1:

```
ASSERT EITHER { PERM network_access } \
OR { PERM send_packet_out }
```

-

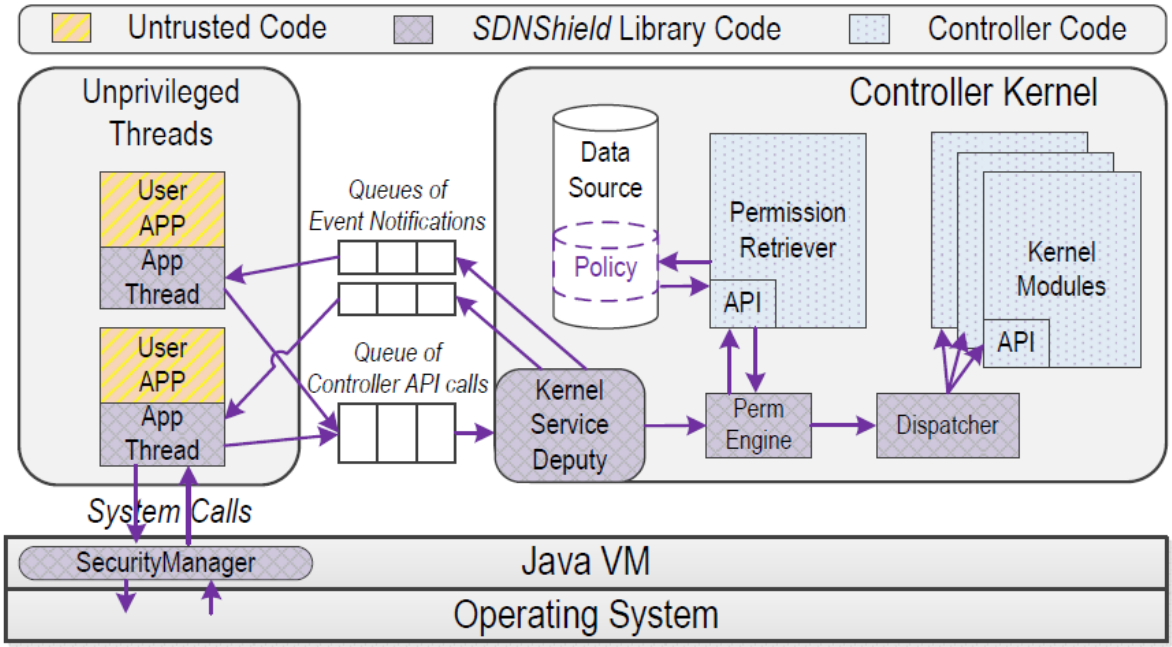
```
PERM visible_topology LIMITING LocalTopo
PERM read_statistics
PERM network_access LIMITING AdminRange
PERM insert_flow
```

```
LET LocalTopo = {SWITCH 0,1... LINK 3,4...}
LET AdminRange = {IP_DST 10.1.0.0 \
MASK 255.255.0.0}
```

permission manifest *security policy*

Isolation Architecture

- Design Goals
  - Execution and memory isolation
  - Mediating syscalls
  - Efficiency
  - No modification to Apps
- Design Choices
  - Thread-level isolation
  - Language-based reference monitoring (Java VM)

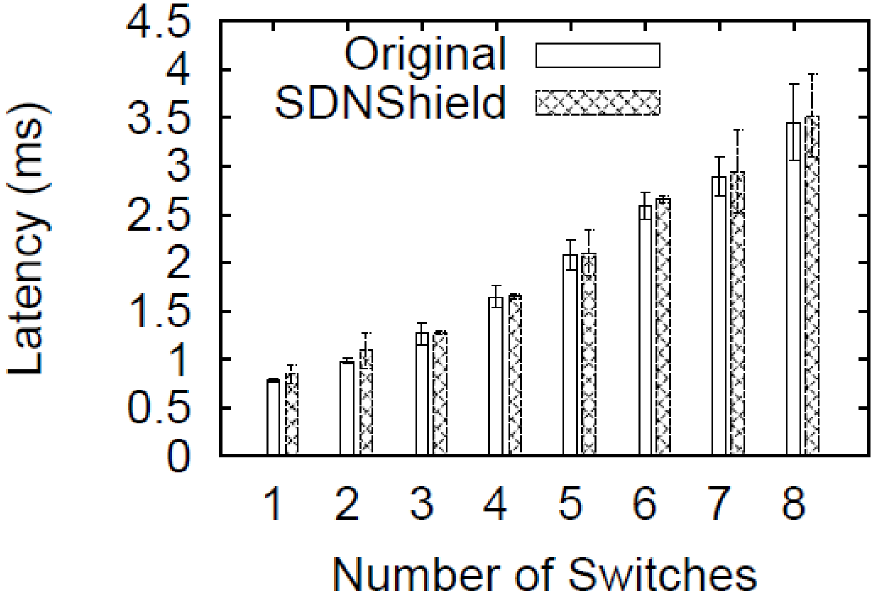


Implementation

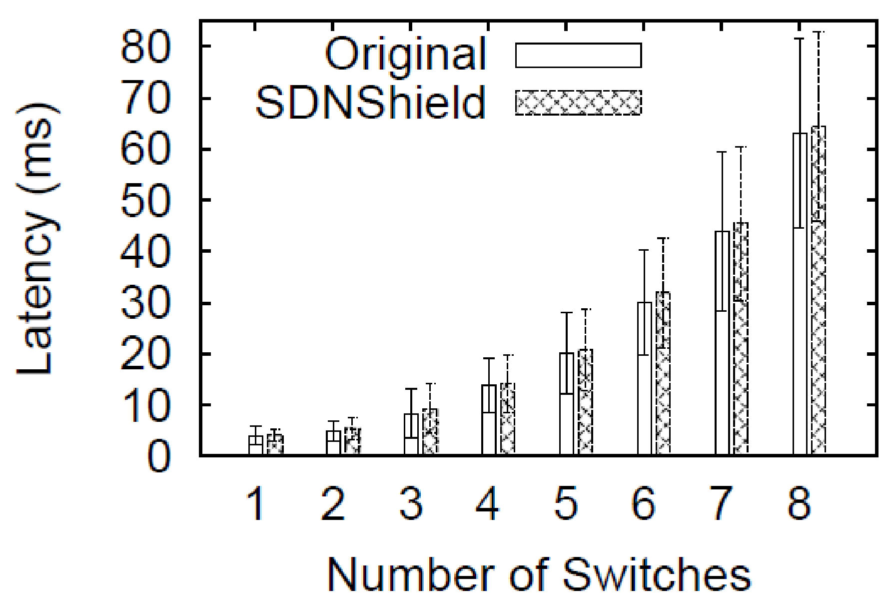
- Platform-independent reconciliation engine and permission engine
  - Parses permission manifests and security policies
  - Library for efficient permission checking
  - 23k lines of code in Java
- Controller extensions
  - Implemented on OpenDaylight and Floodlight
  - Inter-thread communication and API wrapping
  - App initiation process

Evaluation

- Latency impact
  - Tested on two real apps
  - Almost unnoticeable latency difference



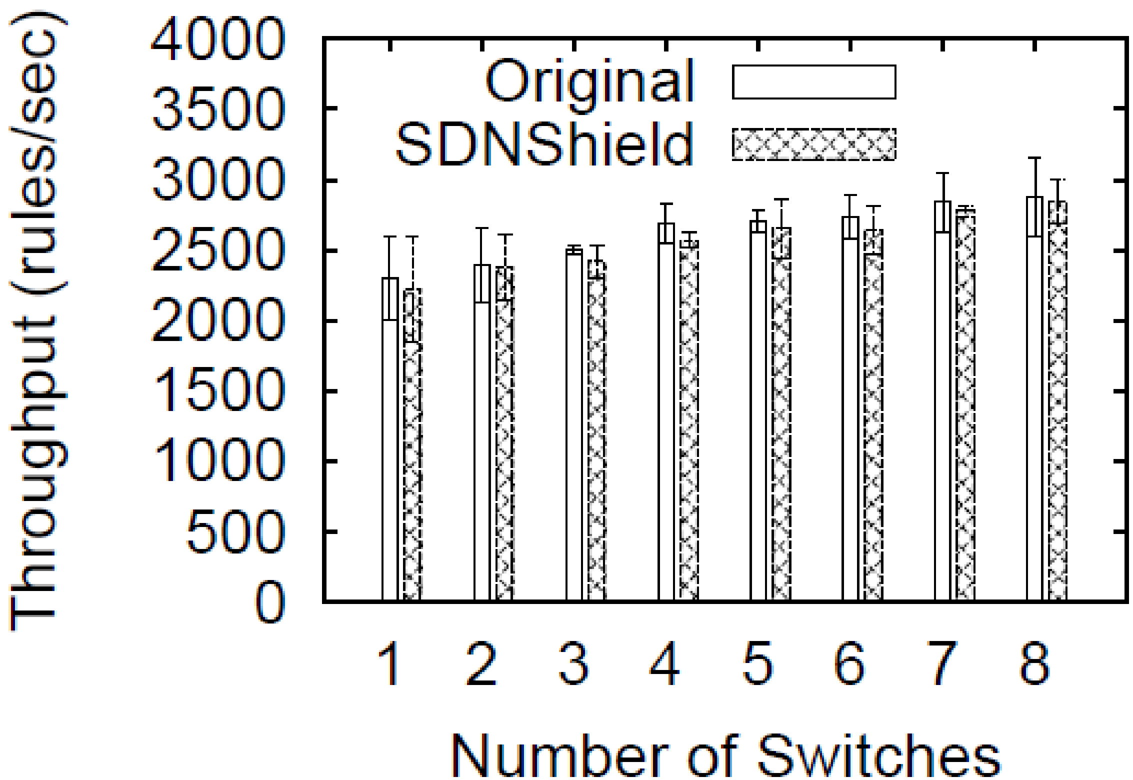
(a) L2 Learning Switch



(b) ALTO TE

Evaluation

- Throughput impact



Roadmap

- SDNShield System Design
- SDNShield Implementation and Evaluations
- Ongoing Extension to REST APIs and NFVs
- Conclusions

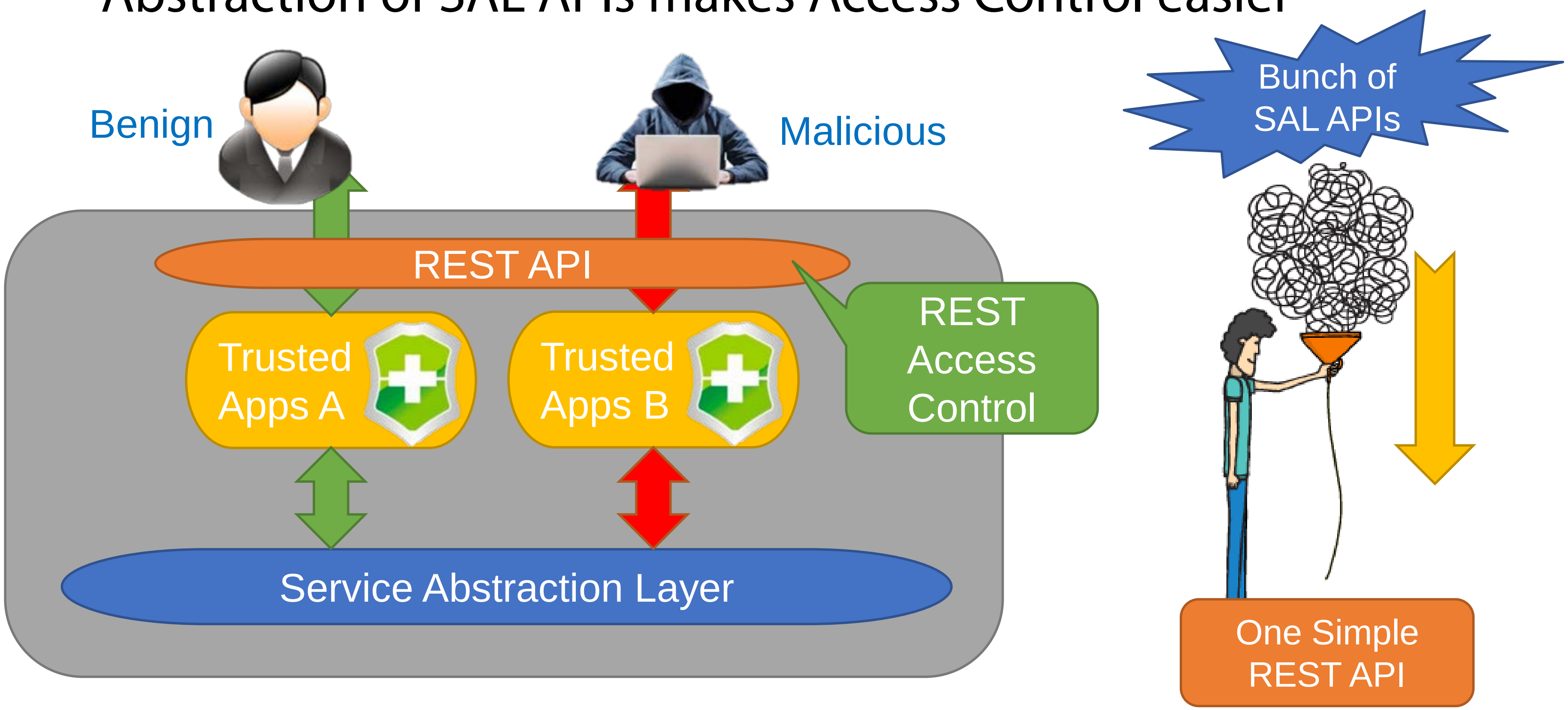
REST APIs

- REST (REpresentational State Transfer)
  - Resource-based: Each URI represents a resource.
  - Client-Server: Client connects to Server via HTTP protocol
  - HTTP verbs: POST-Create, GET-Select, PUT-Update, DELETE-Delete
- Benefits
  - Simple, flexible, unified, scalable
- Example

HTTP Verb	URI	Description
POST	/v2.0/routers	Create a router
DELETE	/v2.0/routers/{router_id}	Delete a router
GET	/v2.0/routers	Query all routers

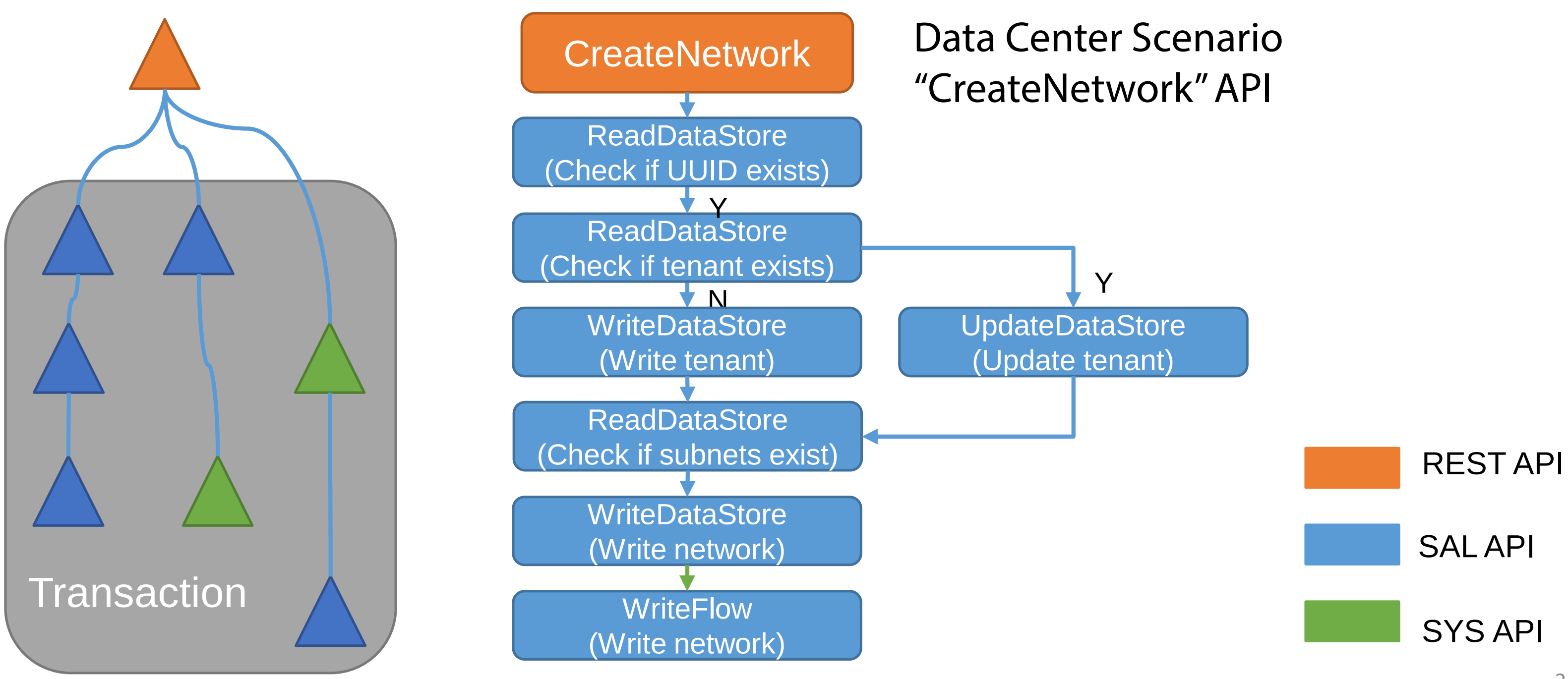
Motivation in REST API Access Control

- Attacks outside controller remain
- Trusted Apps with certain privileges are still dangerous
- Hackers with access to trusted apps can attack underlay network
- Abstraction of SAL APIs makes Access Control easier



Design for REST API Access Control

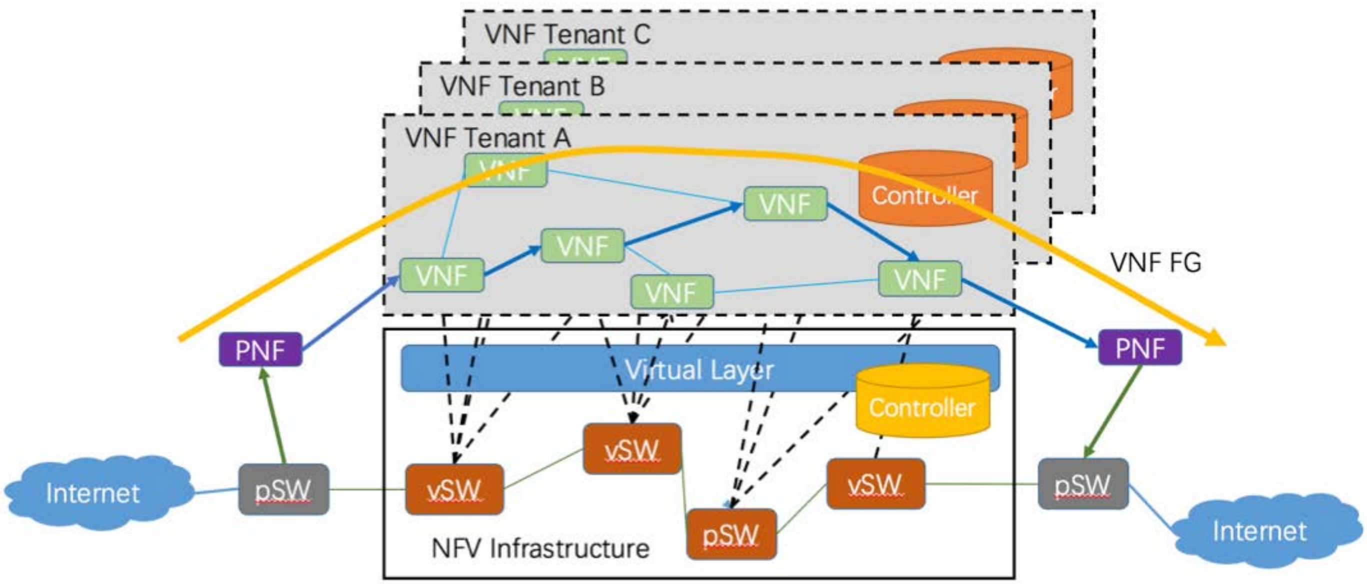
Based on Mapping between REST APIs & SAL APIs
- REST API generally is an independent transaction
- A transaction contains several sequential primitive API calls
- REST API can be mapped into an API tree in time order



NFV Scenarios

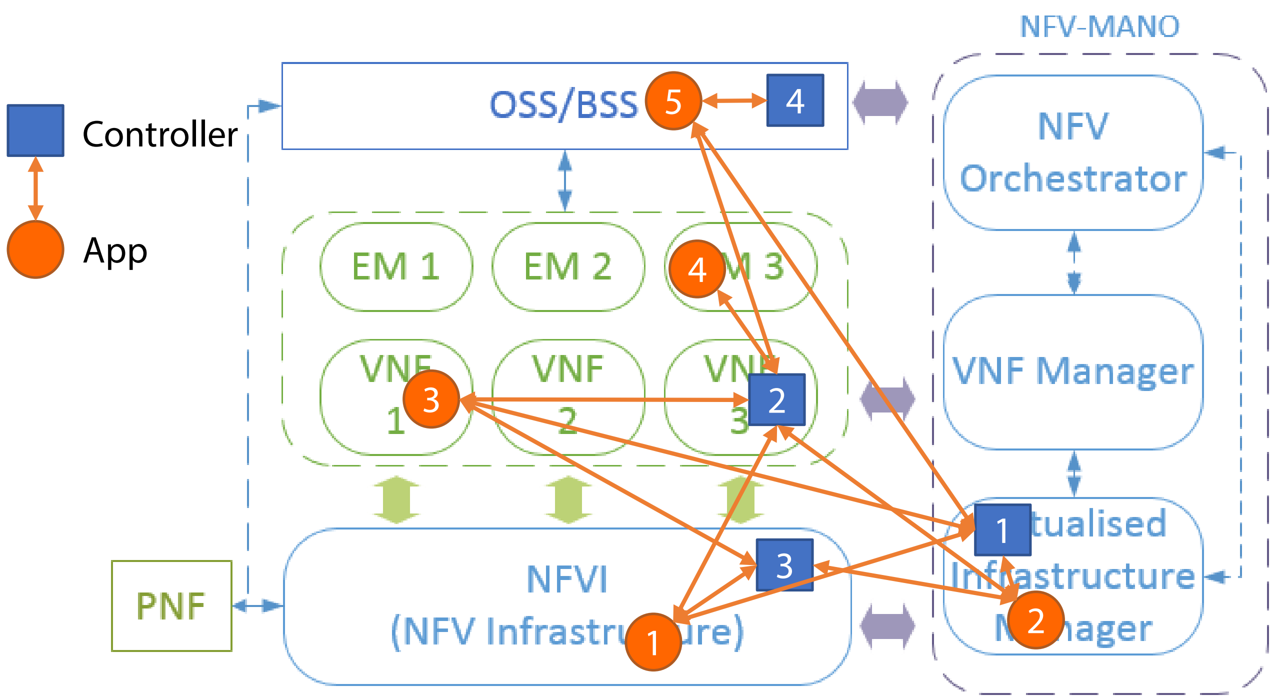
- Network functions are virtualized in cloud based platform.
- Controllers are responsible for leading traffic through a series of network functions, thus providing network services.
- NFVI providers can even provide controllers for tenants to manipulate their traffic on providers' infrastructure.

More important role in network
But very limited security work done



Extending SDNShield to NFVs

- App-Controller Interaction
  - Multiple controllers locate in different layer, each with specific role
  - Cross-layer interactions between Apps and Controllers
- SDNShield in NFV
  - Controller is vital for NFV, (REST) interactions are frequent and complex
  - How to have efficient and effective access control for NFVs



Conclusions

- A novel and flexible permission control system for SDN applications
- Fine-grained permission abstractions
- Limited increase on administration burden

- Extending SDNShield to REST APIs and NFVs

http://list.cs.northwestern.edu/sdn

Comparison

	Control-plane or Data-plane	Allow App Cooperation?	Protection beyond Flow Conflict	Protection beyond CP/DP Channel?
FortNOX/FRESCO	CP	Yes	No	No
FlowVisor	CP	No	Yes	No
AvantGuard	DP	N/A	N/A	N/A
SDNShield	CP	Yes	Yes	Yes

NFV Scenarios

- Network functions are virtualized in cloud based platform.
- Controllers are responsible for leading traffic through a series of network functions, thus providing network services.
- NFVI providers can even provide controllers for tenants to manipulate their traffic on providers' infrastructure.

More important role in network
But limited security works done