NDSS SYMPOSIUM

When Match Fields Do Not Need to Match: Buffered Packet Hijacking in SDN

Jiahao Cao, Renjie Xie, ***Kun Sun***, Qi Li,
Guofei Gu, and Mingwei Xu

Tsinghua University
George Mason University
Texas A&M University

Outline

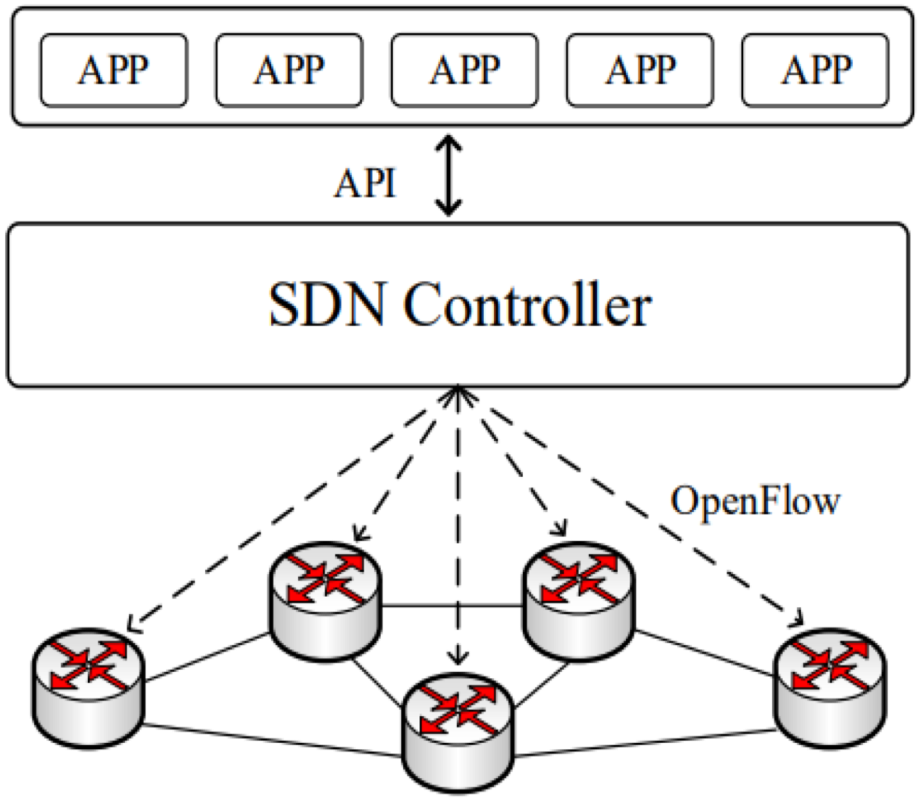
- SDN Overview
- Background on SDN Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

Outline

- SDN Overview
- Background on Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

SDN Overview

- SDN applications (apps)
  - Extend controller capacities and SDN functionalities
- SDN controller
  - Take centralized network control
- SDN switches
  - Forward and process flows according to the controller

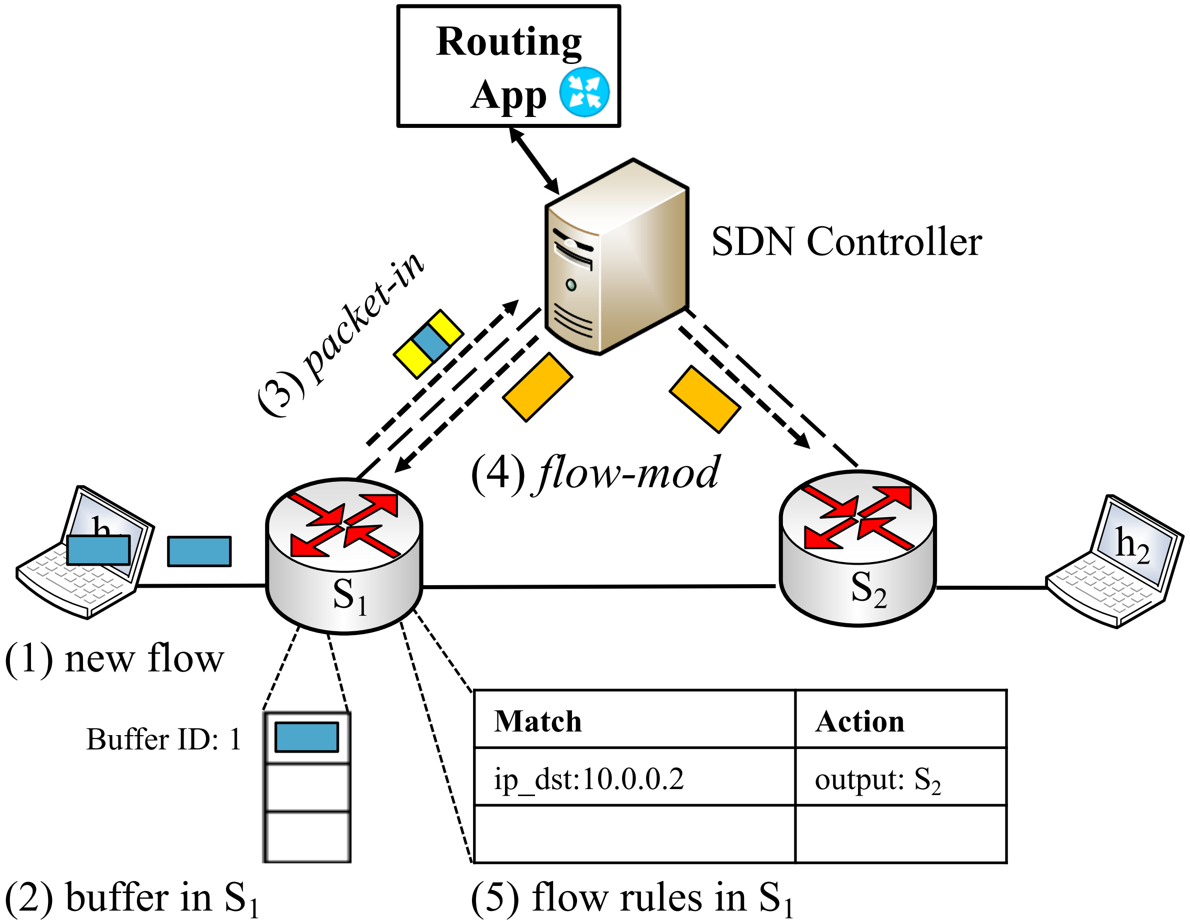


Outline

- SDN Overview
- Background on SDN Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

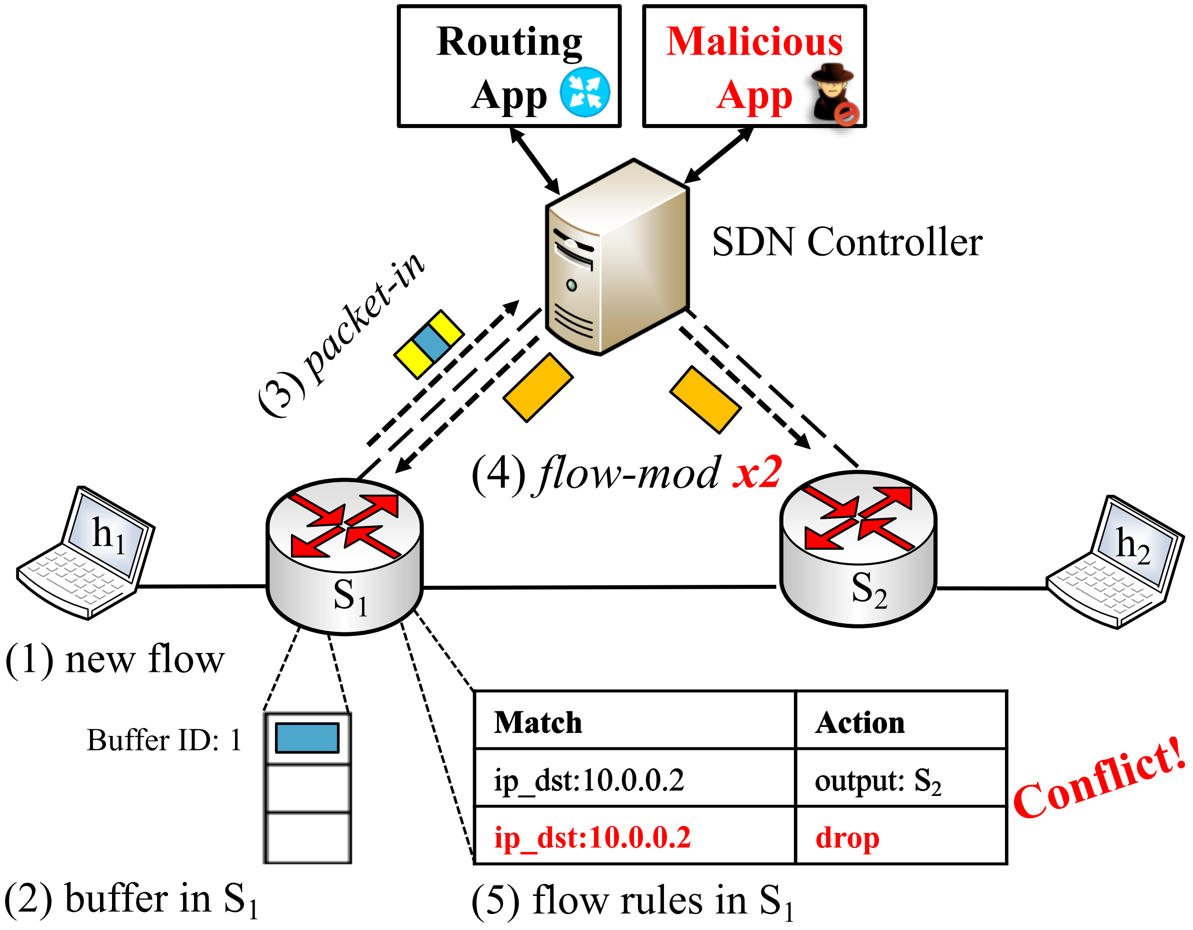
Rule Installation in SDN

- Packet-in
  - Query network decisions for a new flow
  - Contain a **buffer ID** and **packet headers**
- Flow-mod
  1. Install rules with **match fields** and **actions**
  2. Specify a **buffer ID** to release a buffered packet



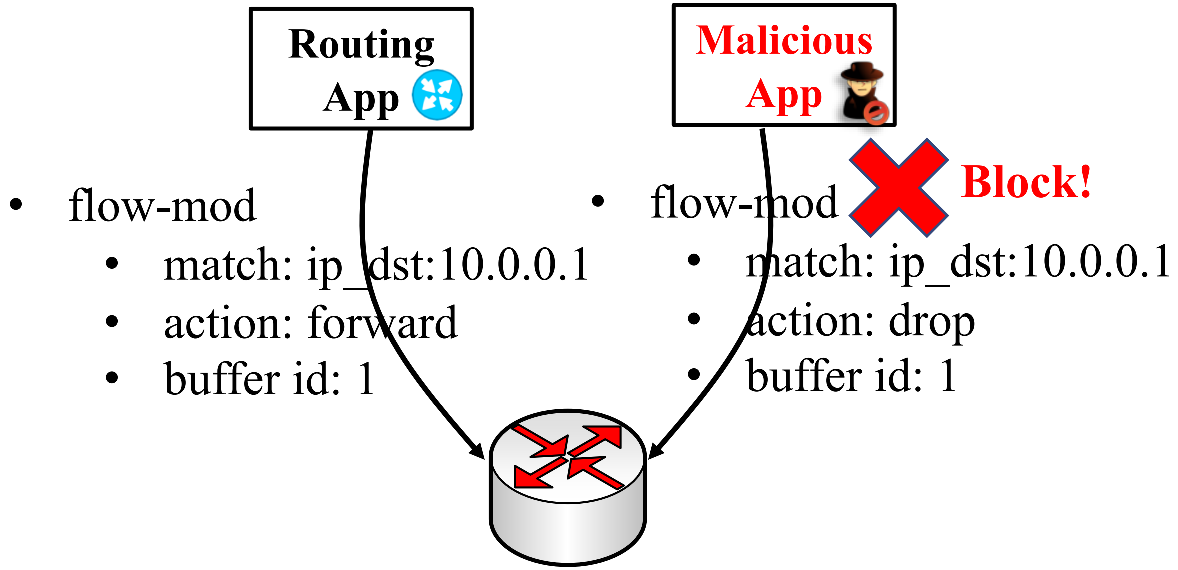
Rule Conflict in SDN

- Conflict reason
  - **Multiple apps** process the same flow may generate conflicting rules
- Conflict abuse
  - Apps install conflicting rules to **override other apps' decisions**



Rule Conflict Detection

- Rule conflict detection
  - Extract **match fields** and **actions** in all flow-mod messages
  - Check potential conflict when installing new rules



VerfiFlow (NSDI ’13), SE-Floodlight (NDSS ‘15), FortNOX (HotSDN ‘12)…

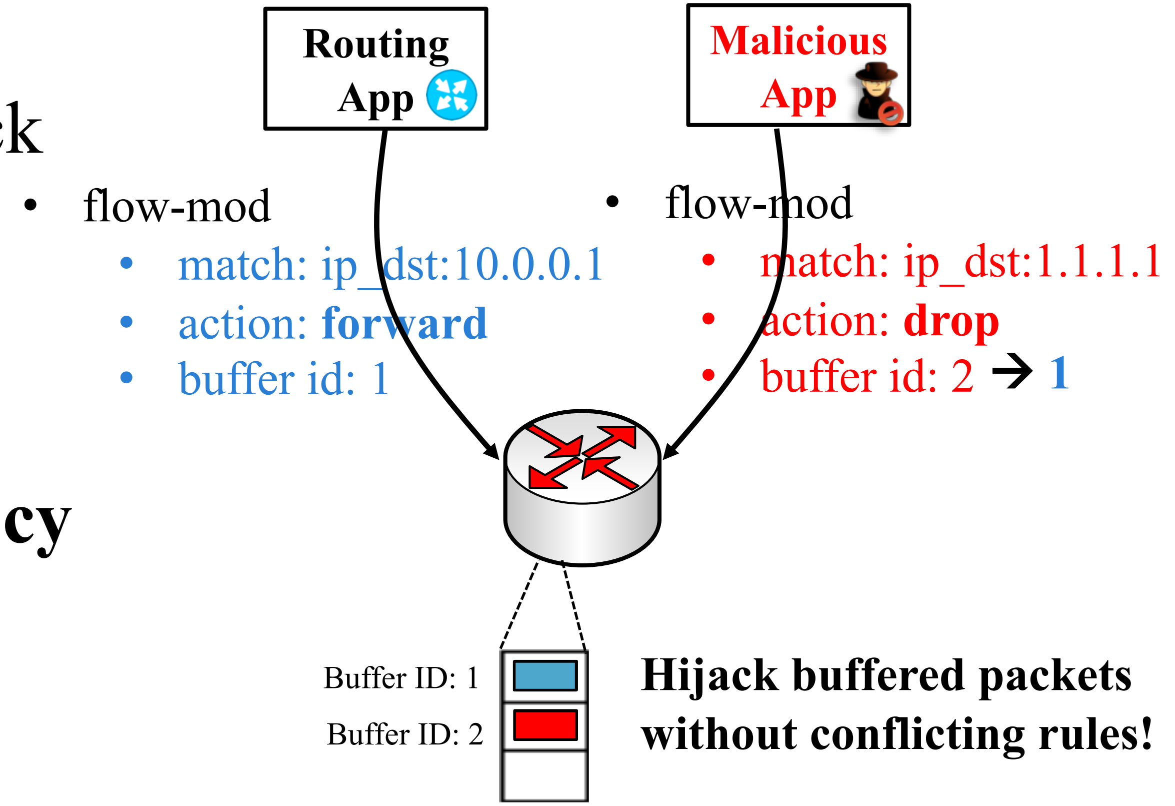
Do not consider potential **buffer ID abuse**

Outline

- SDN Overview
- Background on SDN Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

Buffered Packet Hijacking Vulnerability

- Mechanism
  - **Manipulate buffer IDs** to hijack buffered packets
- Root Cause
  - No checking on the **inconsistency** between buffer IDs and match fields when installing rules



Outline

- SDN Overview
- Background on SDN Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

Threat Model

- Attacker Objective
  - Exploit the vulnerability to attack **all three SDN layers**
- System Assumptions
  - SDN controllers, switches, and control channels are secure
  - Existing SDN defense may be deployed
  - Apps are **untrusted**, which may originate from third parties
  - A malicious app has **basic permissions** of listening packet-in and installing flow rules

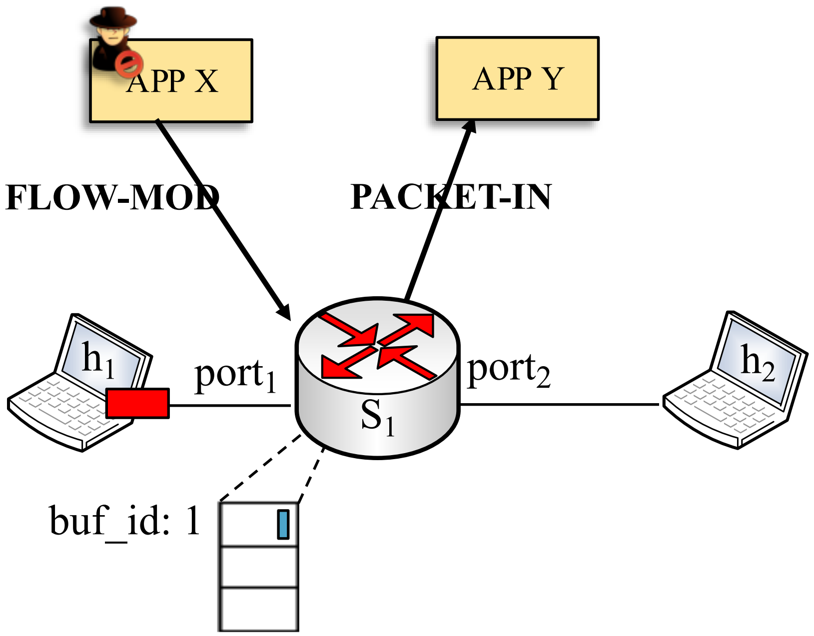
Attacks and Testbed

- Four attacks
  - Attacking application
    1. cross-app poisoning
  - Attacking control plane
    2. control traffic amplification
  - Attacking data plane
    3. security policy bypass
    4. TCP connection disruption

- Real SDN testbed
  - Open source controller
    - Floodlight
  - Commercial SDN switches
    - EdgeCore AS4610-54T
  - Real background flows
    - Traffic trace from CAIDA
    - Crafted test flows

Attack 1: Cross-App Poisoning (CAP)

- A malicious app resends modified buffered packets to the controller



APP Y learns:
(Host, Port) = (h_1, $port_1$)

APP Y learns:
(Host, Port) = (h_2, $port_1$)

Incorrect mapping!

APP X: FLOW-MOD
- match: other flow
- buf_id: 1
- action: set-field (IP_SRC→IP_h_2), output:controller

Evading Defense against CAP

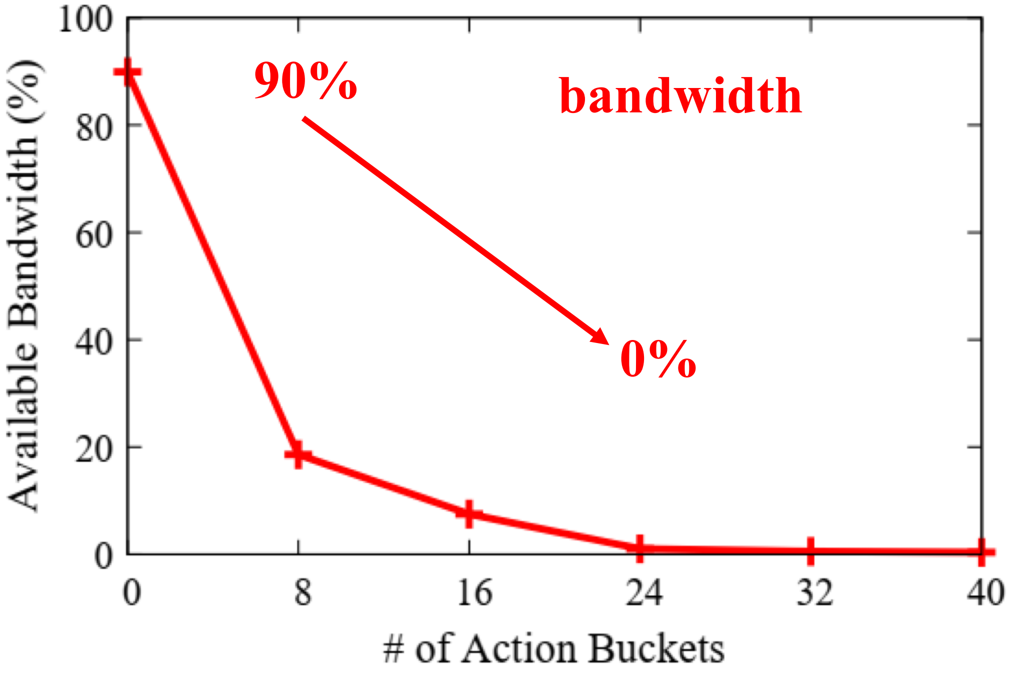
- Existing CAP attacks and defense
  - Attack by modifying **shared data objects** in the **control plane**
  - Defend by checking information flow control policy violations*

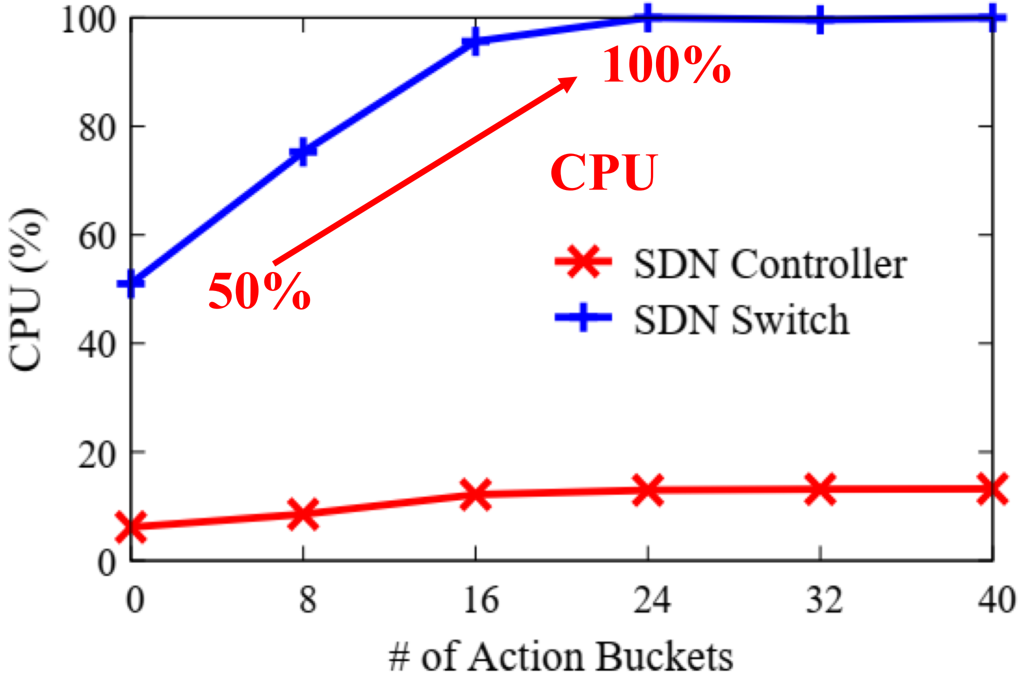
- This CAP attack
  - Manipulate **buffered packets** in the **data plane**
  - Evade defense since there are **no policy violations**

* Ujcich, Benjamin E., et al. "Cross-app poisoning in software-defined networking." CCS '18

Attack 2: Control Traffic Amplification Bomb

- A malicious app copies massive buffered packets to trigger packet-in messages consuming bandwidth and computing resources





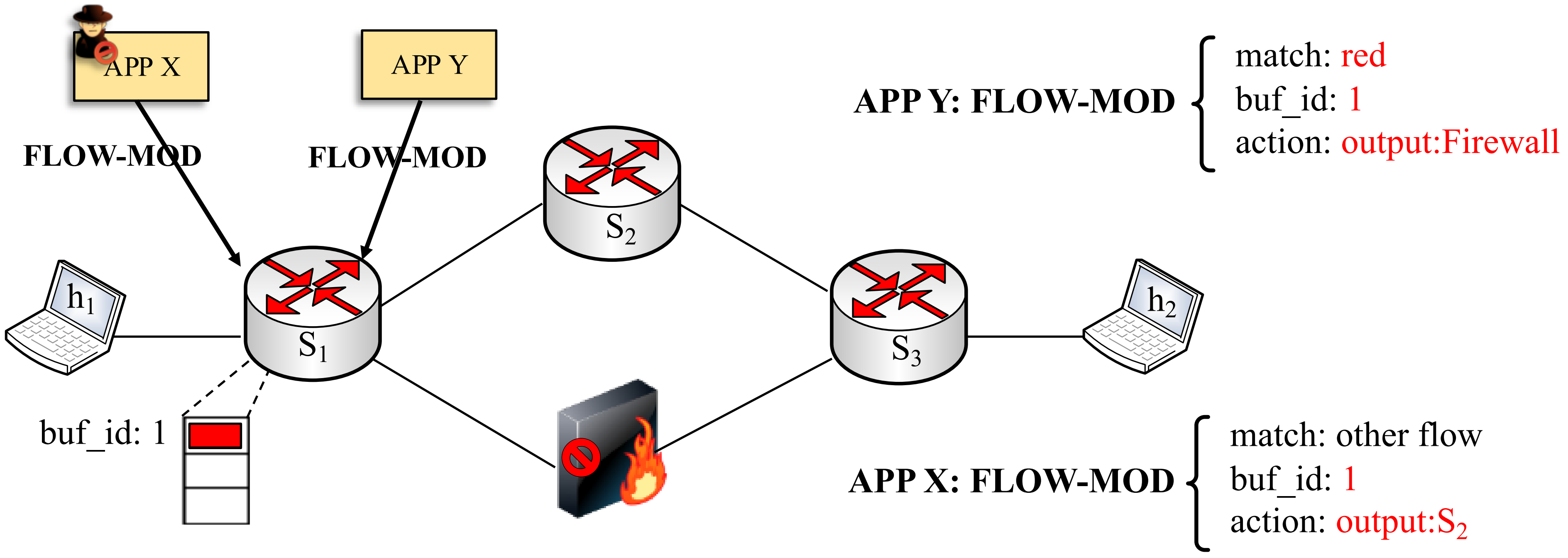
Evading Defense against Packet-in Flooding

- Existing flooding attacks and defense
  - Attack by generating packets **matching no rules** to trigger massive packet-in messages
  - Detect malicious flows or adopt TCP SYN proxy to throttle TCP-based flooding*
- This flooding attack
  - Hijack **buffered packets** of benign flows to trigger massive packet-in messages
  - Generate **no malicious flows** and can hijack **UDP flows**

- Shin, Seungwon, et al. "Avant-guard: Scalable and vigilant switch flow management in software-defined networks." CCS '13
  Shang, Gao, et al. "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks." *INFOCOM '17*

Attack 3: Network Security Policy Bypass

- A malicious app redirects buffered packets to different ports



Successfully bypass firewall !

Evading Defense against Security Bypass

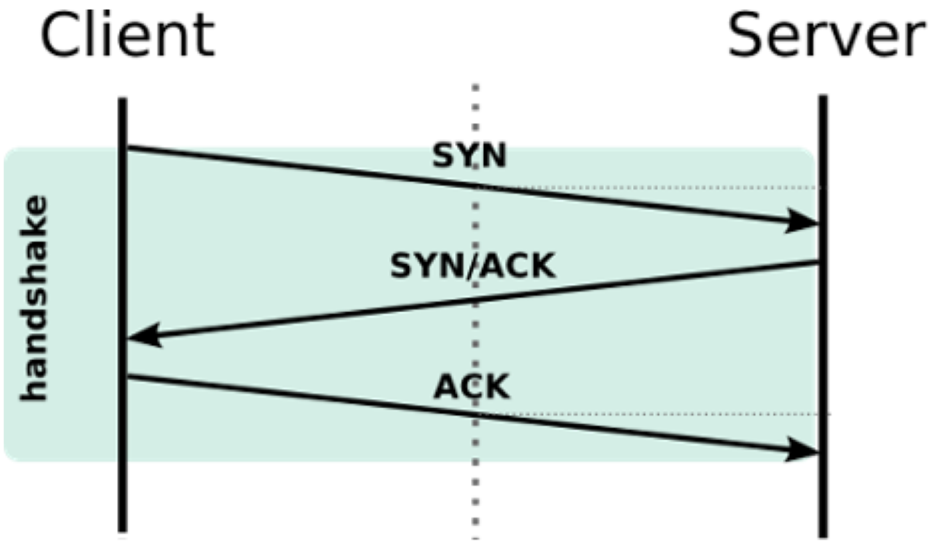
- Existing security bypass attacks and defense
  - Generate **conflicting rules** to bypass security policies
  - Detect rule conflict to prevent security policy bypass*

- This attack
  - Manipulate **buffer IDs** to bypass security policies
  - Evade defense by generating **no conflicting rules**

* Porras, Phillip A., et al. "Securing the software defined network control layer." NDSS '15.
Khurshid, Ahmed, et al. "Veriflow: Verifying network-wide invariants in real time." NSDI '13
Porras, Philip, et al. "A security enforcement kernel for OpenFlow networks." HotSDN '12

Attack 4: TCP Connection Disruption

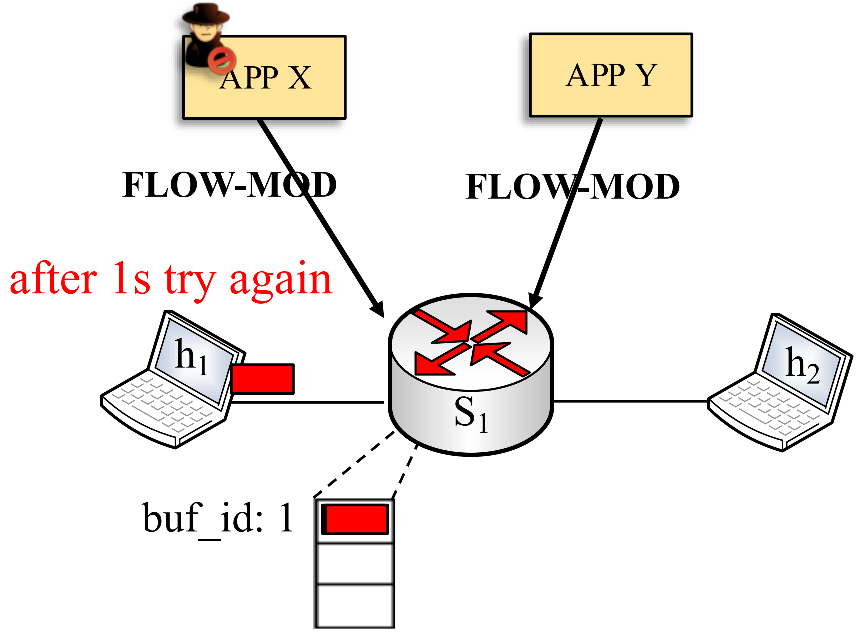
- TCP three-way handshake process
  - A TCP connection is established only after a successful TCP three-way handshake

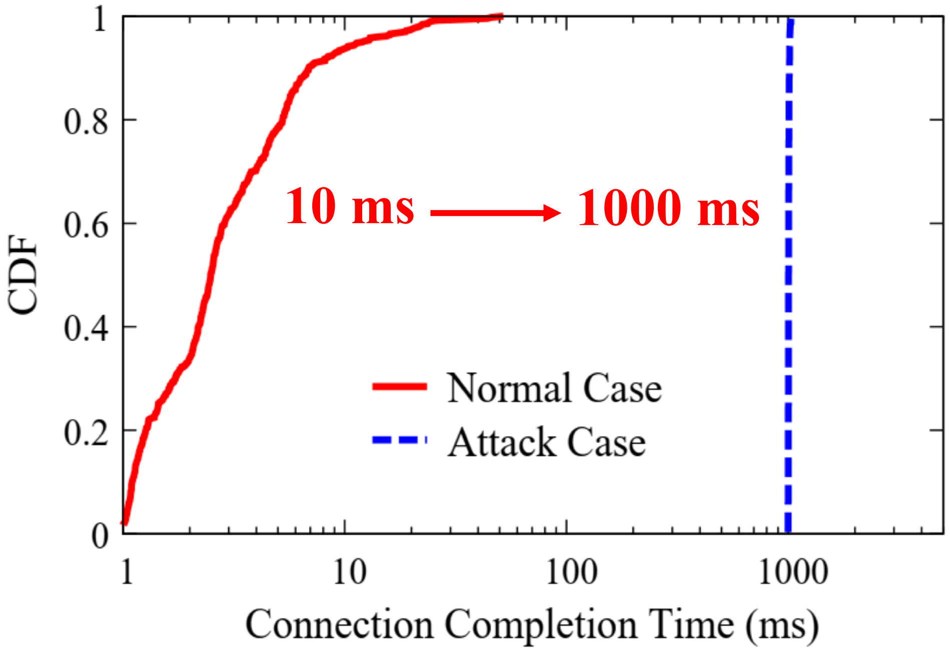


The first packet of a TCP flow is always the TCP SYN packet

Attack 4: TCP Connection Disruption

- A malicious app drops a buffered TCP SYN packet



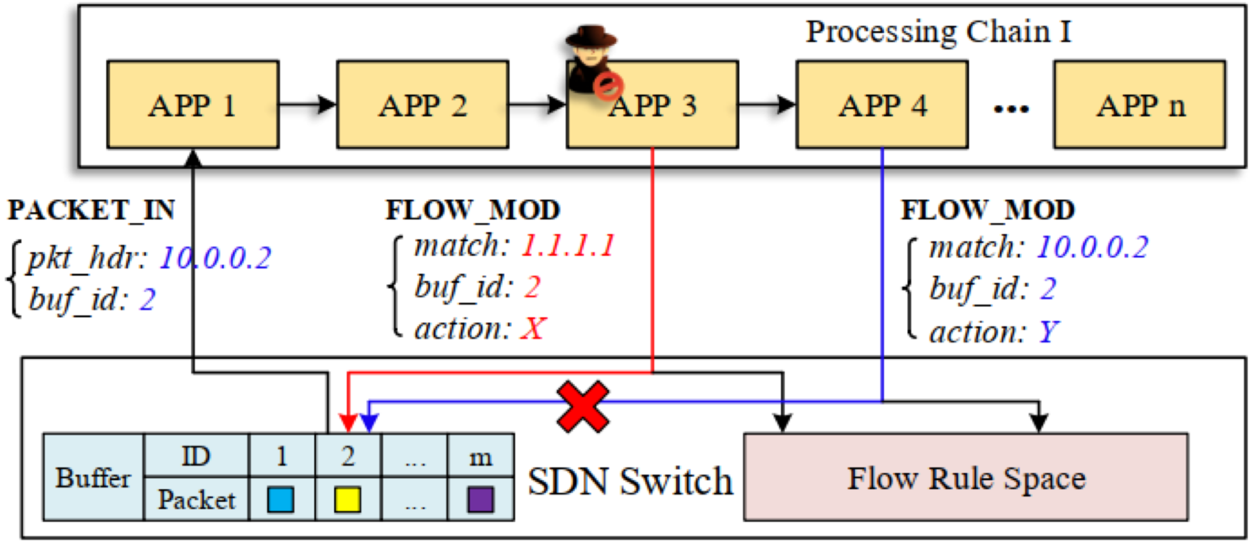


Every 100 *ms* latency may cost 1% in business revenue for Amazon.
No existing SDN defense solutions consider this attack !

Hijacking Probability: **Intra-Chain** Hijacking

- Single Processing Chain
  - Apps in the same processing chain process packet-in and send flow-mod messages in turn

- Success Condition
  - A malicious app is **in front of** the app that will process the flow (target app)

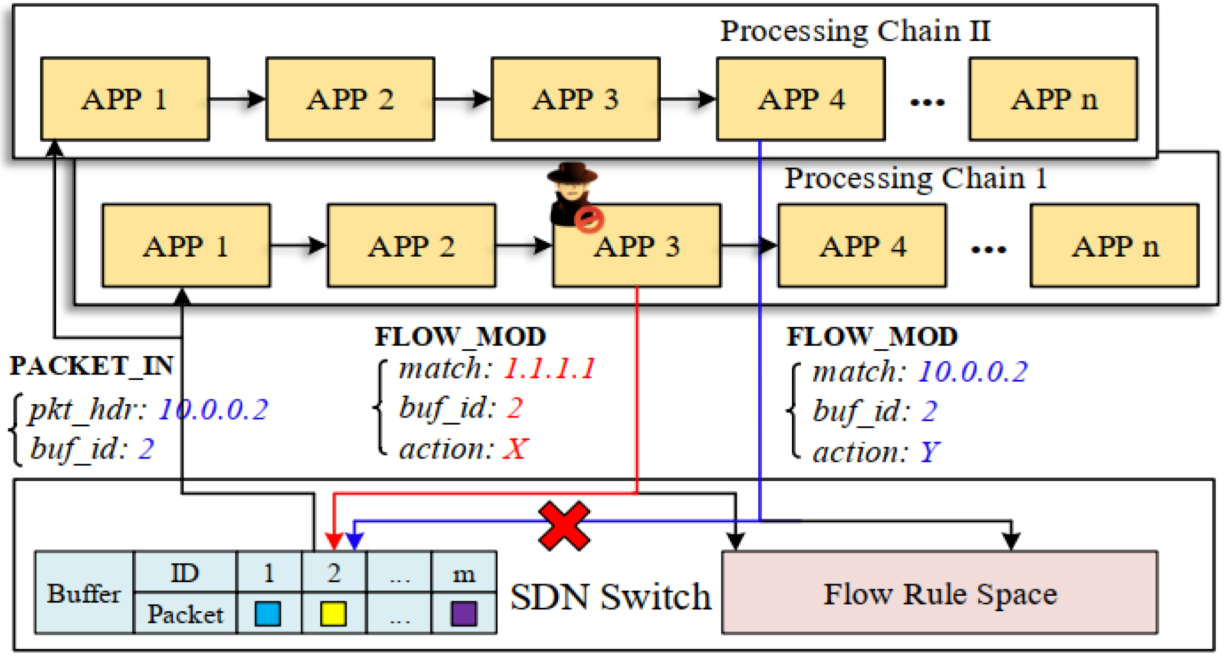


Hijacking Probability: **Inter-Chain** Hijacking

- Multiple Processing Chains
  - Apps in different processing chains process packet-in and send flow-mod messages independently

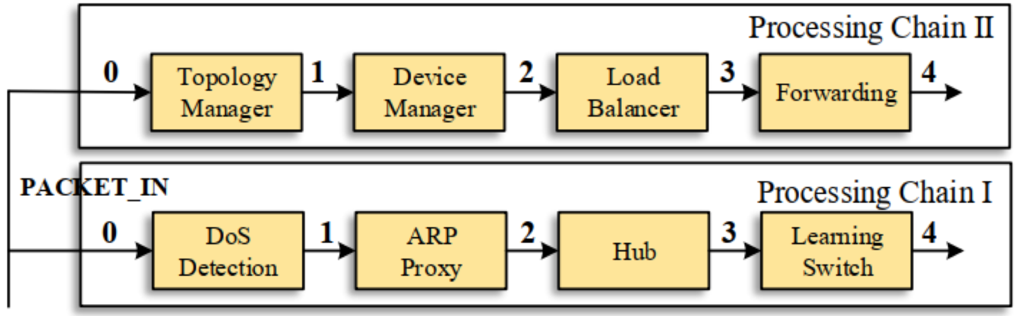
- Success Condition
  - A malicious app could be **in any position,** if

$$\sum_{i=1}^{malicious} delay < \sum_{i=1}^{target} delay$$



Hijacking Probability: Experimental Results

- Experiments with two processing chains in real SDN testbed



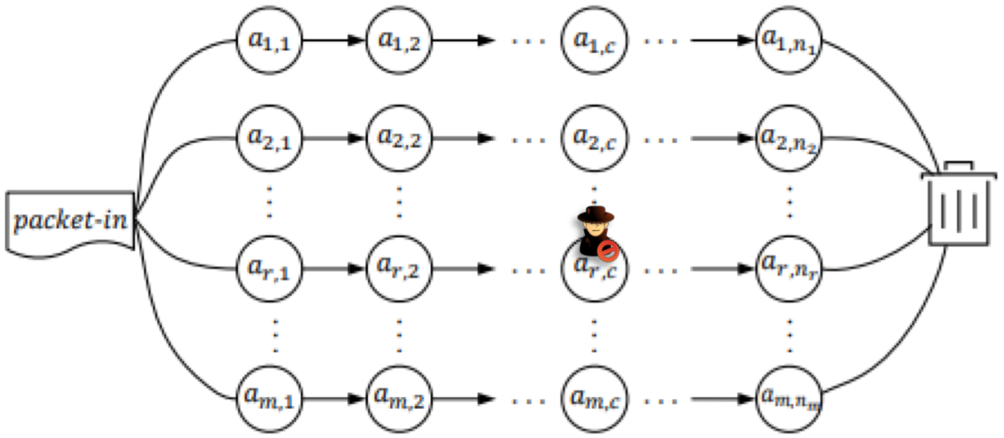
- Intra-chain hijacking probability is either **0** or **100%**
- Inter-chain hijacking probability **decreases** when the malicious app moves towards tail, e.g., from **100%** to **36.3%** for Load Balancer

Malicious App's Position	Hijacking Probability with a Target App				
	DoS Detection	Hub	Learning Switch	Load Balancer	Forwarding
Chain I: 0	100.0%	100.0%	100.0%	100.0%	100.0%
Chain I: 1	0	100.0%	100.0%	90.0%	91.7%
Chain I: 2	0	100.0%	100.0%	70.5%	82.0%
Chain I: 3	0	0	100.0%	68.5%	80.9%
Chain I: 4	0	0	0	36.3%	57.1%
Note	**Intra-Chain Hijacking**			**Inter-Chain Hijacking**	

Malicious App's Position	Hijacking Probability with a Target App				
	Load Balancer	Forwarding	DoS Detection	Hub	Learning Switch
Chain II: 0	100.0%	100.0%	89.3%	100.0%	100.0%
Chain II: 1	100.0%	100.0%	48.8%	92.2%	95.7%
Chain II: 2	100.0%	100.0%	33.3%	85.7%	93.9%
Chain II: 3	0	100.0%	9.7%	30.6%	62.3%
Chain II: 4	0	0	8.3%	18.3%	41.9%
Note	**Intra-Chain Hijacking**		**Inter-Chain Hijacking**		

Hijacking Probability: Theory Analysis

- Derive hijacking probability from processing chain model



- $a_{r,c}$: malicious app, the c-th application in the r-th processing chain
- $a_{i,j}$: target app, the i-th application in the j-th processing chain
- $f_{i,j}$: probability density function of processing delays in $a_{i,j}$

- Intra-chain hijacking probability:

$$p_{intra}(a_{r,c}, a_{r,j}) = \begin{cases} 100\%, & if\ j \in \{1, 2, ..., c-1\} \\ 0, & if\ j \in \{c+1, c+2, ..., n_r\} \end{cases}$$

- Inter-chain hijacking probability:

$$p_{inter}(a_{r,c}, a_{i,j}) = \int_{-\infty}^{0} \underbrace{\left(\int_{-\infty}^{+\infty} \cdots \int_{-\infty}^{+\infty}\right)}_{j+c-1} \prod_{k=1}^{j} f_{i,k}(t_k - t_{k-1}) \cdot$$

$$\prod_{k=1}^{c-1} f_{r,k}(t_{j+k-1} - t_{j+k}) \cdot f_{r,c}(t_{j+c-1} - z) \cdot \prod_{k=1}^{j+c-1} dt_k \cdot dz$$

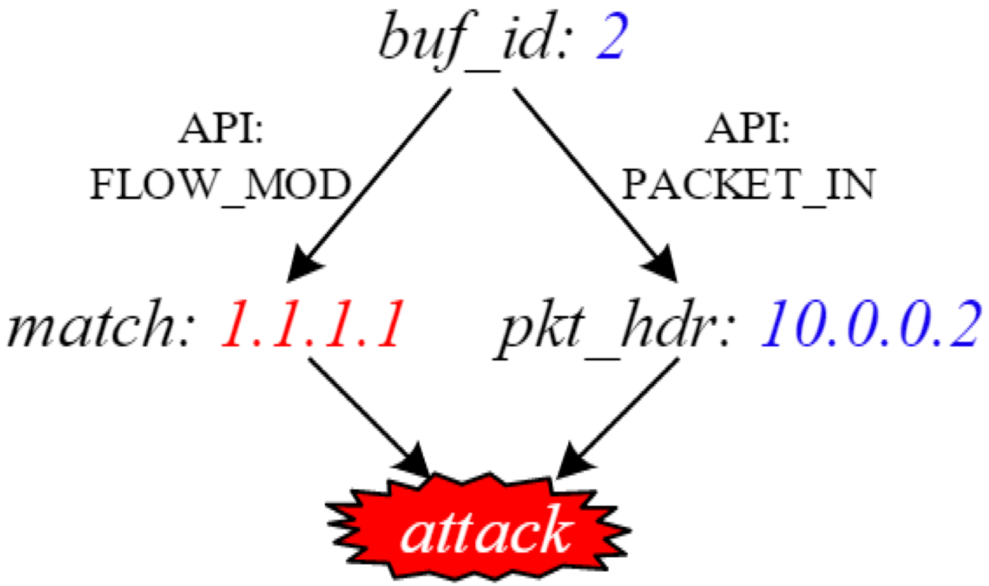
Details in our paper!

Outline

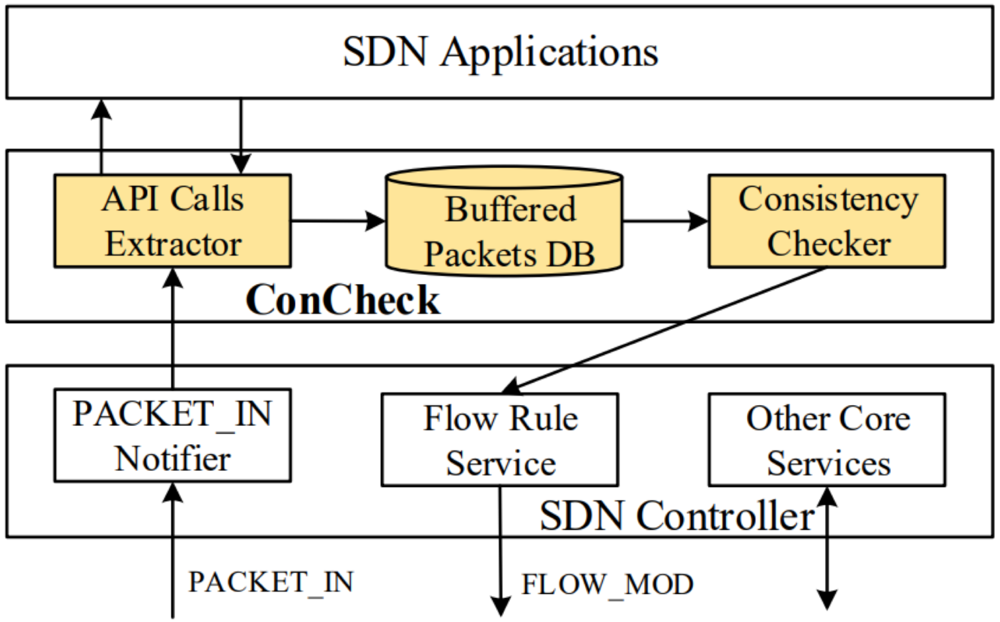
- SDN Overview
- Background on Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

Defense: ConCheck

- Add **consistency check** between buffer IDs and match fields



Detection Example



ConCheck Architecture

Outline

- SDN Overview
- Background on Rule Installation
- A New Vulnerability: Buffered Packet Hijacking
- Buffered Packet Hijacking Attacks
- Defense
- Conclusion

Conclusion

- We discover a new vulnerability in SDN rule installation.
- We identify four buffered packet hijacking attacks that disrupt all SDN layers and can evade all existing defense systems.
- We propose a lightweight and application-transparent countermeasure.

Thank you!

Kun Sun
ksun3@gmu.edu

Backup: Permissions

- The ratio of applications with the permission of listening packet-in messages and installing flow rules

Controller	Total APPs	APPs with the Permission	Ratio
`OpenDaylight Neon`†	13	6	46.2%
`ONOS v2.1.0-rc1`	97	23	23.7%
`Floodlight v1.2`	29	12	41.4%
`RYU v4.31`	28	19	67.9%
`POX eel version`	18	11	61.1%

† Only counting the applications implemented with *openflowplugin*.

Many apps have the permissions

Backup: Vulnerability Report & Response

- Mainstream SDN vendor Pica8
  - Acknowledged our report and said "we have filed tracking tickets and are waiting for product management decision on releasing the fix in major/minor or patch builds"
- Mainstream carrier-grade SDN controller ONOS
  - Helped us file a defect in the ONOS community with the comment that "the defect will be visible to the community and this info can be available for someone to pick it up to fix it"
- Popular SDN controller RYU
  - Several developers and users in the community confirmed our report

Evaluation on ConCheck

- We implement a prototype of ConCheck in Floodlight