Stateful Processing in SDN

Presented by Weijie Liu and Imani Palmer

CS563/ECE524 Advanced Computer Security

University of Illinois at Urbana-Champaign

Outline

- Overview of SDN and OpenState
- Two Applications on OpenState
- Experiments and Results
- Future Work and Conclusion

Overview

- Software-Defined Networking
  - Data plane dumb
  - “Smartness” at the controller
  - Convenience and flexibility

- Makes building stateful applications difficult
  - Ex: Stateful Firewall
  - Need explicit involvement of the controller
    - Extra signaling load and processing delay

- Can we offload the work load of stateful processing from the controller onto forwarding devices in some extent?

Difficulty of Stateful Processing

- Challenge: Stateful Processing
- Limited access to packet-level information
  - Difficult to support stateful packet inspection
- Implement stateful processing logic in controller
  - Keep the switches simple
- Require lots of processing and memory space in controller side

OpenState

- Supports stateful per flow processing
- Creates an abstraction that generalizes OpenFlow rules as a eXtended Finite State Machines (XFSM)
- APIs on Ryu using flow tables in switches to represent XFSM
- Retain the benefits of stateful processing with OpenState while taking advantage of SDN distributed capabilities

[1] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch” ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 44–51, 2014.

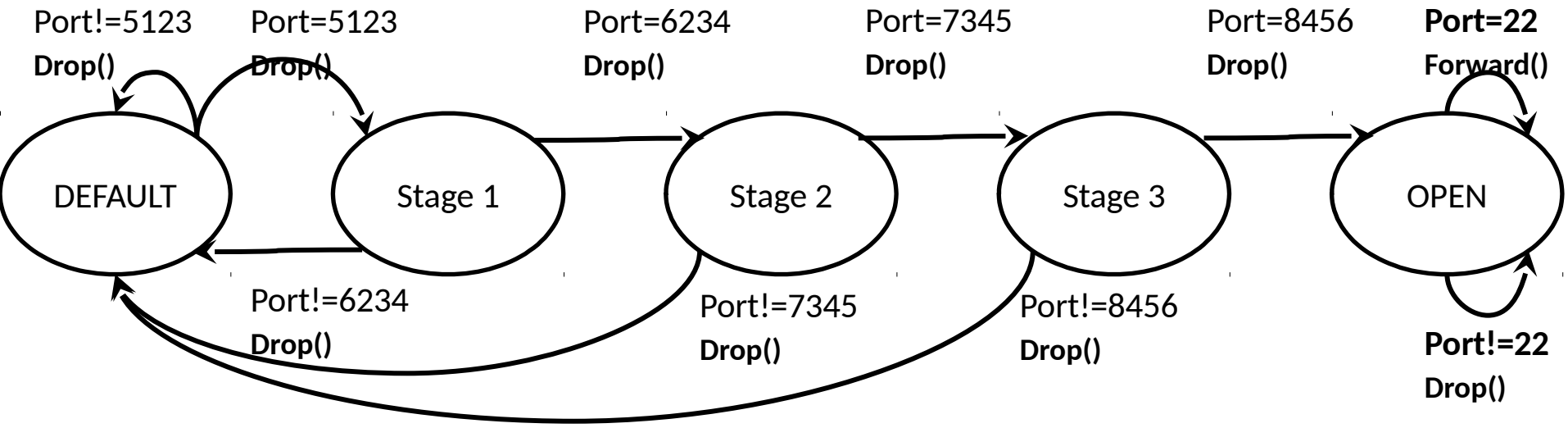
Our contributions

- Two applications on OpenState:
  Distributed Port Knocking (DPK)
  Per-flow Update Consistency (PUC)

- Experiments

Distributed Port Knocking (DPK)

Port Knocking can be represented as a state machine:



State Table

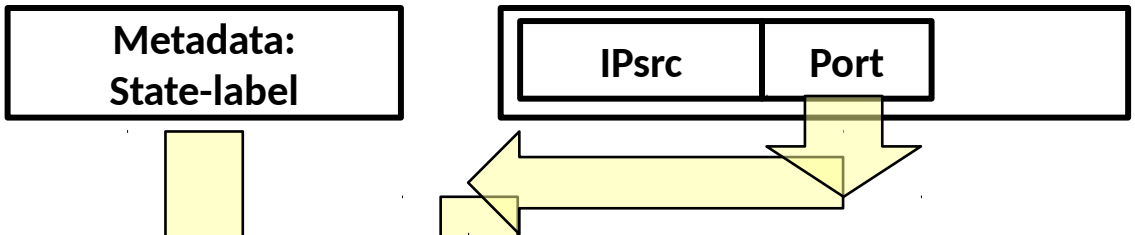
- State Table: recording current state of a certain flow

Flow key	state
IPsr=.. ..	… … ..
IPsrc=1.2.3.4	STAGE-3
IPsrc=5.6.7.8	OPEN
IPsrc=… …	… … ..
IPsrc=… …	… … ..
IPsrc=no match	DEFAULT

XFSM Table

MATCH: <state, port> → ACTION: <drop/forward, state_transition>

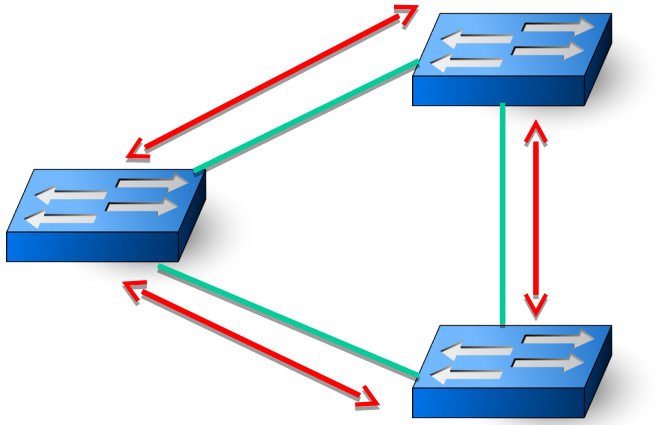
- XFSM Table: represent the state machine



Match fields		Actions	
state	event	action	Next-state
DEFAULT	Port=5123	drop	STAGE-1
STAGE-1	Port=6234	drop	STAGE-2
STAGE-2	Port=7345	drop	STAGE-3
STAGE-3	Port=8456	drop	OPEN
OPEN	Port=22	forward	OPEN
OPEN	Port=*	drop	OPEN
*	Port=*	drop	DEFAULT

Synchronization of Switches

- A valid port knocking might pass through different switches
- How to share states among switches?
- Solution: When the state of a switch changes because of a certain packet, transmit it to other switches.



Per-Flow Update Consistency (PUC)

- Old and new configurations co-exist during network updates
- Per-flow Update Consistency: each flow processed by the old configuration or the new, but NOT the mixture of the two
- Per-packet Update Consistency is implemented in [1] but per-flow update consistency is not
- OpenState: an excellent fit for Per-flow update consistency

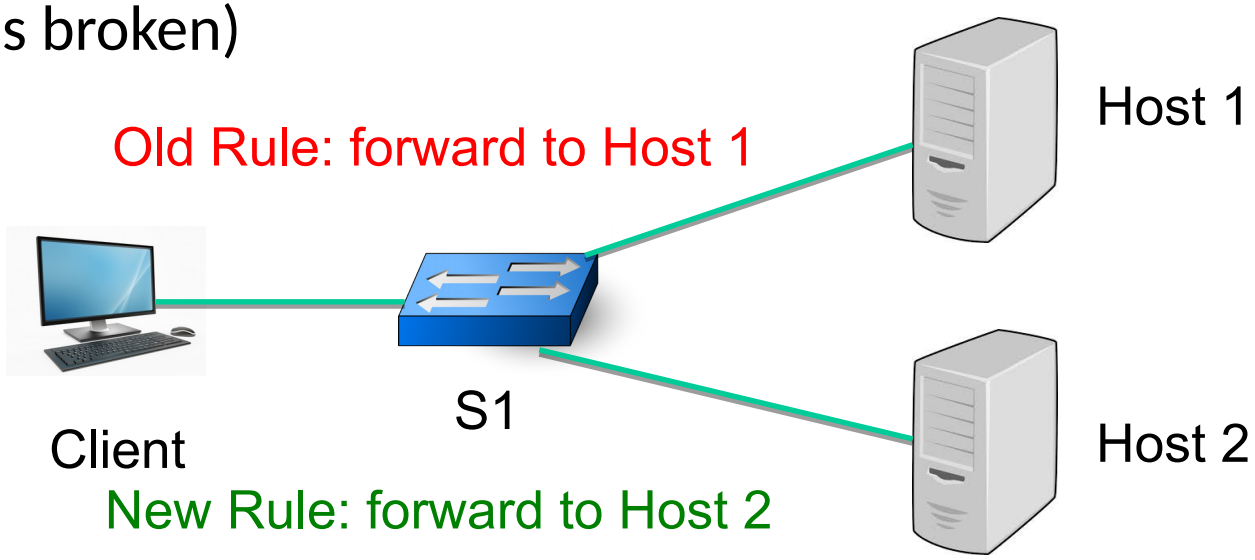
[2] Reitblatt, Mark, et al. "Abstractions for network update." Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication. ACM, 2012.

Flow Update Inconsistency

- Example:

S1 forwards flows to Host 1; after an network update, S1 forwards flows to Host 2. What if the update happens during the transmission progress of one flow?

(results: part of the flow is to Host 1 and other to Host 2; section is broken)



OpenState enforces Consistency

- Solution with OpenState: using states to differentiate flows before update and those after update

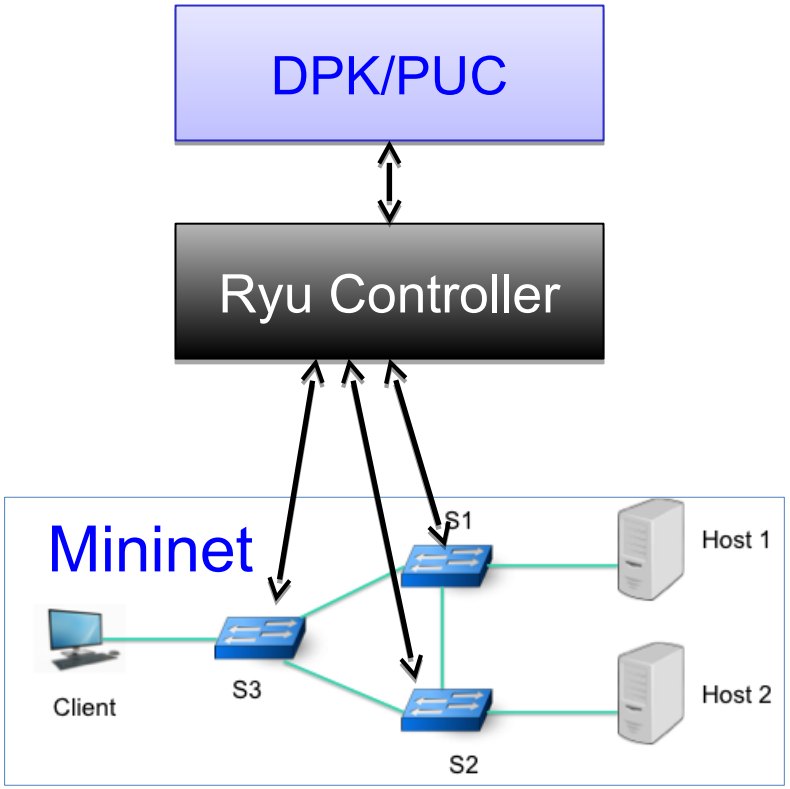
State	Action
State 1	Forward to Host 1
For a new flow w/o state	Set to State 1

Forwarding Rules Update

State	Action
State 2	Forward to Host 2
State 1	Forward to Host 1
For a new flow w/o state	Set to State 2

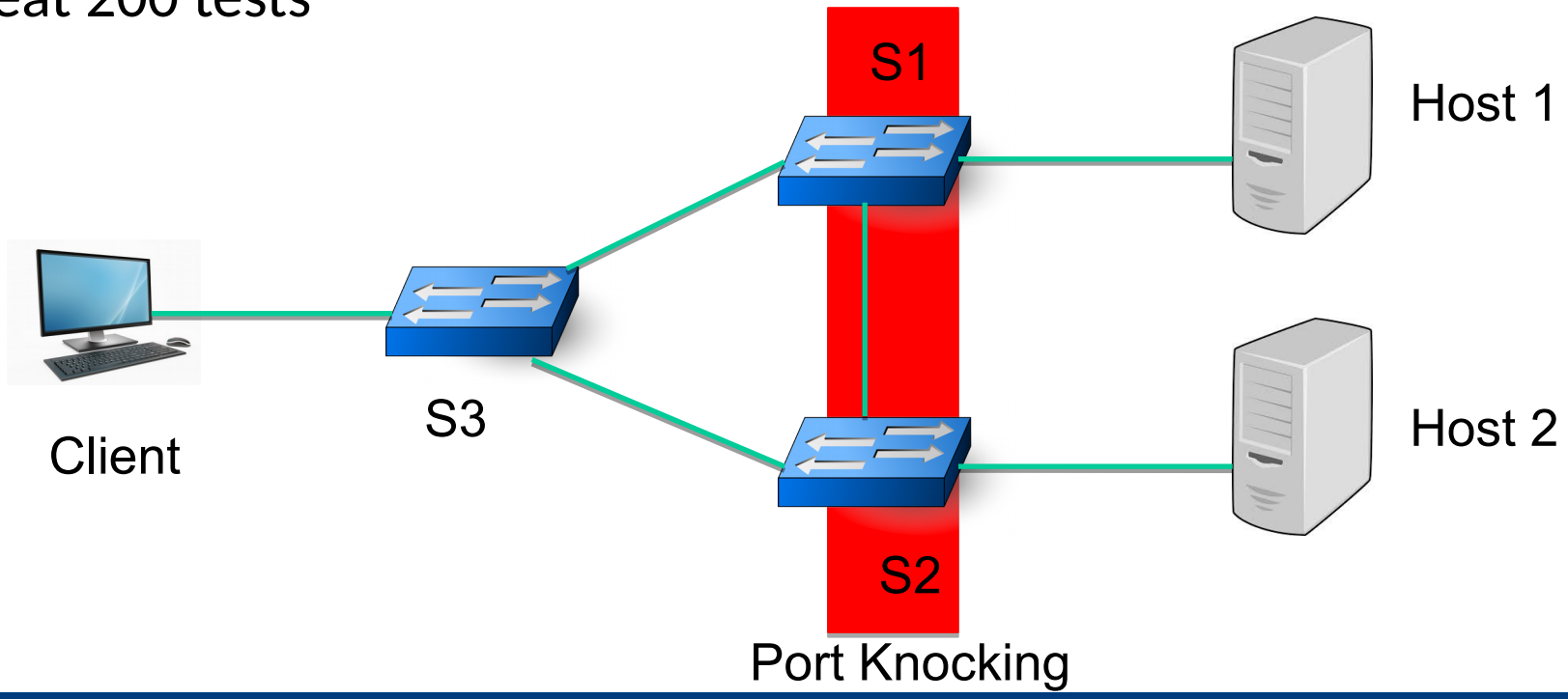
Experiments Setting

- SDN Network: Customized in Mininet v2.1.0
- Controller: Ryu v3.15
- Port Knocking/Update consistency: each more than 250 lines of Python with Ryu & OpenState APIs



DPK Experiment Process

- Assume client knows port numbers, but not the correct order. Client sends out a random permutation of the ports
- DPK XFSM installed in S1 and S2
- S3 forwards packets to S1 and S2 alternately
- Repeat 200 tests



Results: DPK works

- For different lengths of port sequence, percentages of successful port knocking match theoretical calculation.

Correct Port Sequence	Theoretical Successful Rate
[5123 2000]	1/1! = 100%
[5123 6234 2000]	1/2! = 50%
[5123 6234 7345 2000]	1/3! = 16.7%
[5123 6234 7345 8456 2000]	1/4! = 4.2%

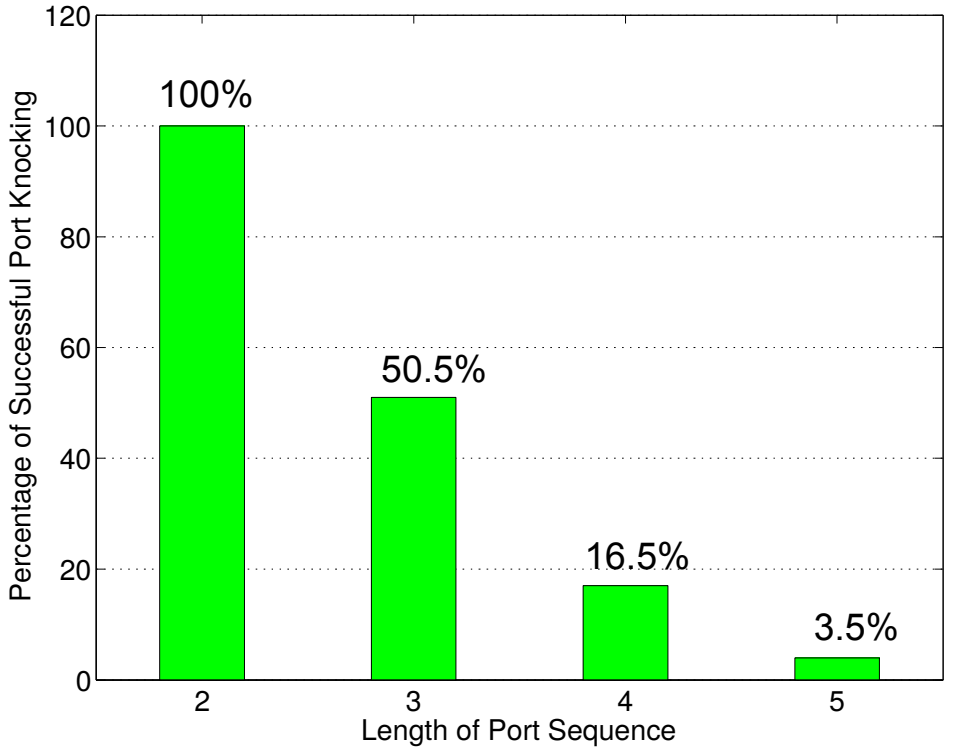


Figure 1. Successful DPK on different lengths of port sequence

Results: small overheads

- Two kinds of overheads: (1) rules installed in switches; (2) packets transmitted between switches

Correct Port Sequence	Avg. # of Pkts btw switches per test
[5123 2000]	1
[5123 6234 2000]	1.5
[5123 6234 7345 2000]	1.66
[5123 6234 7345 8456 2000]	1.55

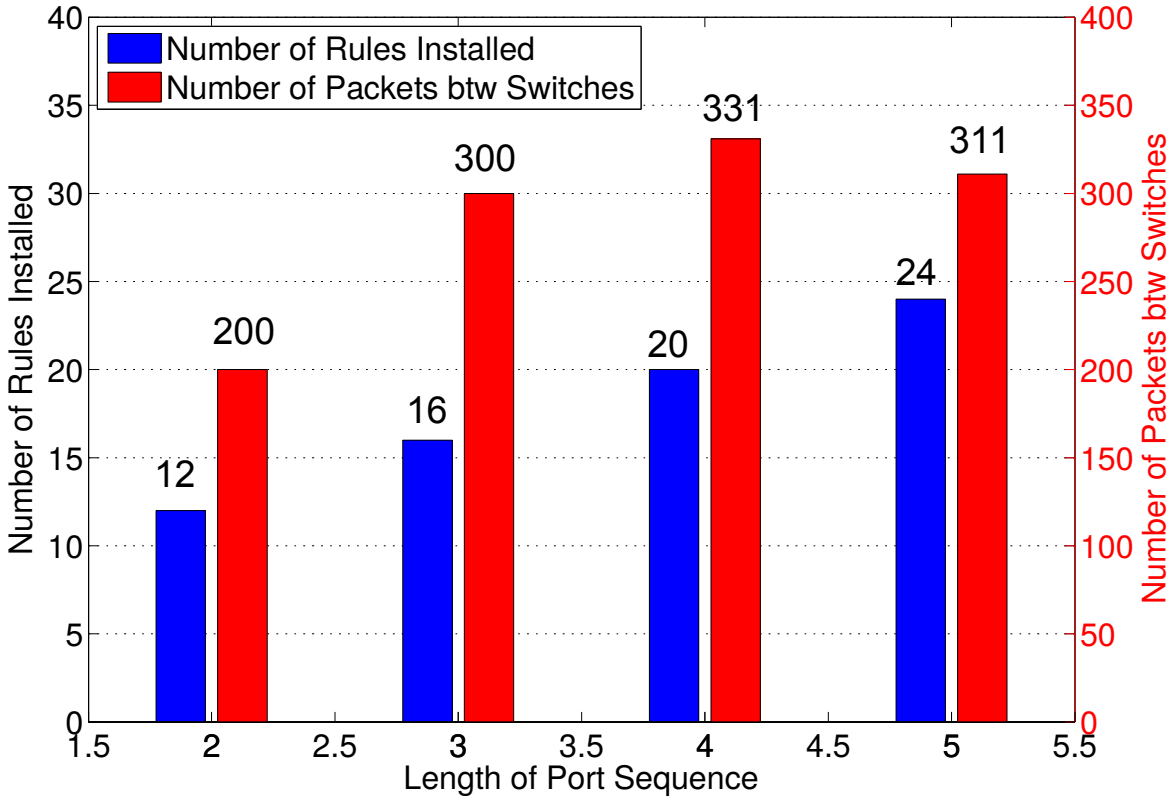


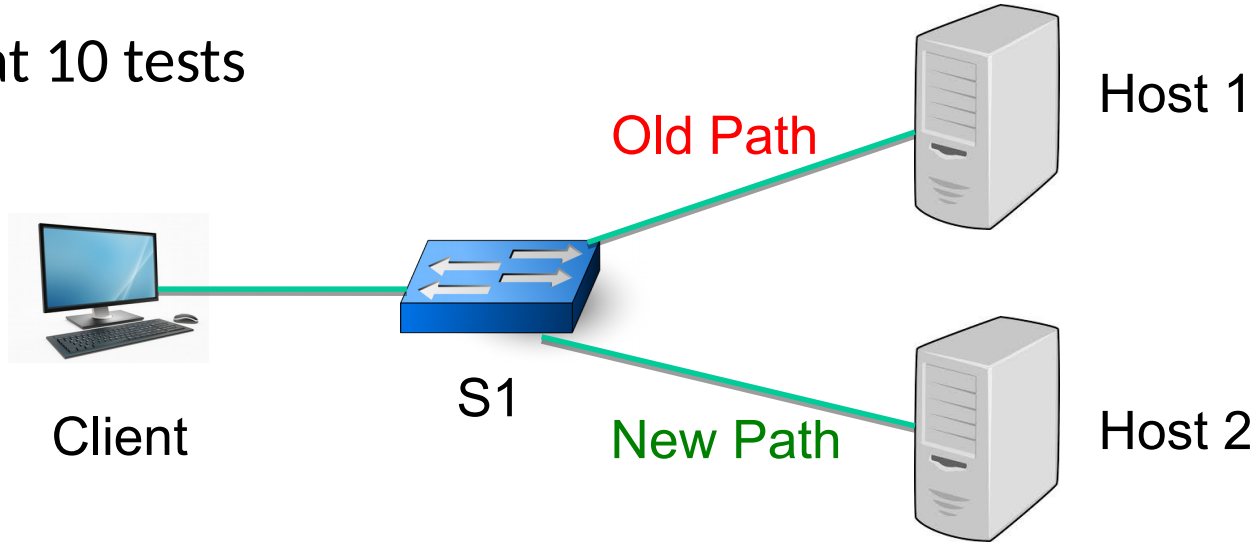
Figure 2. Overheads on different lengths of port sequence

PUC Experiment Process

- A simple update:
  S1 forwards packets to Host 1; after 30 seconds, S1 is updated so that it forwards packets to Host 2

- Client sends a flow (10 bytes ps) for 1 minute

- Repeat 10 tests



Results: PUC works

- With OpenState, Client's flow was still forwarded to Host 1 after the update ⎵ flow's consistency guaranteed
- Without OpenState, controller directly changed the rule in S1: More than half of the flow was forwarded to Host 2.
- But more rules installed with OpenState

	with OpenState	w/o OpenSate
Inconsistent Pkts (%)	0	56
Num of Rules installed	6	3

Table 3. Compare updates with and w/o OpenState

Conclusion & Future Work

- Reducing burden of the controller & distributed stateful processing on SDN are possible

- More Measurements to evaluate the performance of stateful processing
  - Processing delay caused by OpenState
  - Transmission delay of control messages
  - Scalability
- Develop an SDN test API to perform valid performance tests
- Comparison of native control-side method, native XFSM method, and distributed XFSM method

Thank you