

Timing SDN Control Planes to Infer Network Configurations

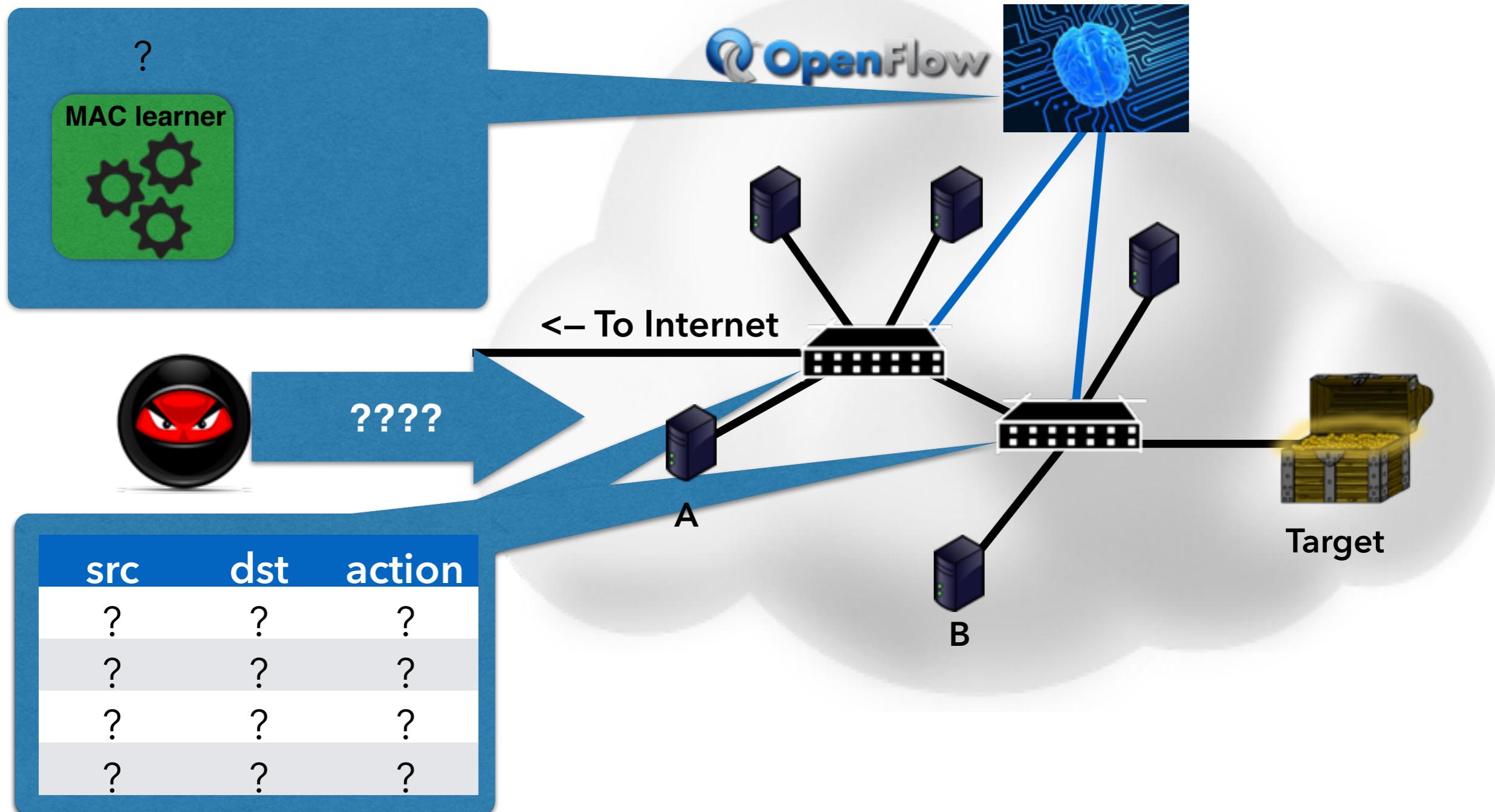
John Sonchack, Adam J. Aviv and Eric Keller



University of Colorado
Boulder



Attack Goal: Probe An OpenFlow Network to Learn its Configuration



Outline

Introduction

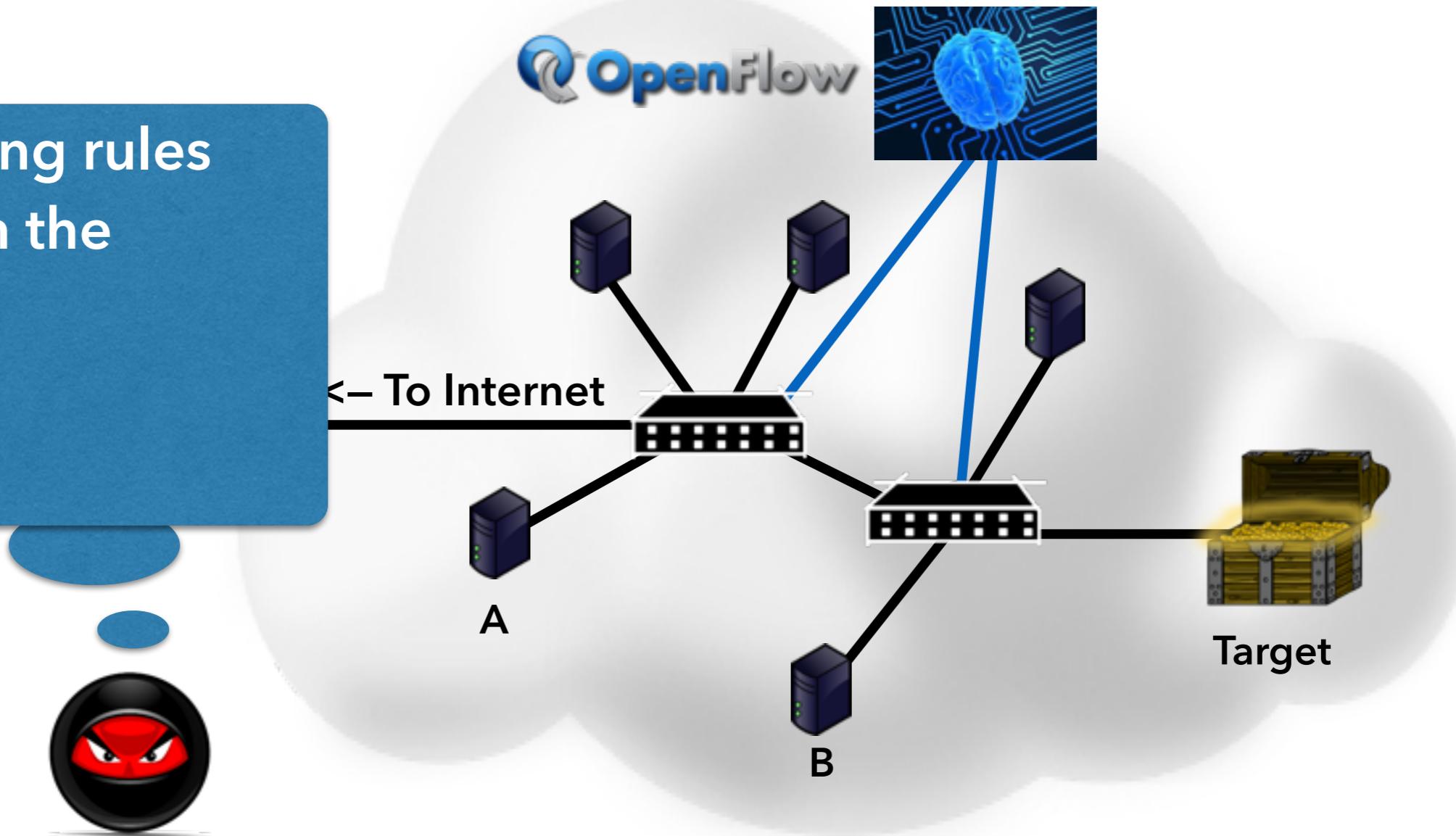
OpenFlow Timing Attacks

A More General OpenFlow
Timing Attack

Preliminary Results

Motivation: Plan Multi-Staged Attacks

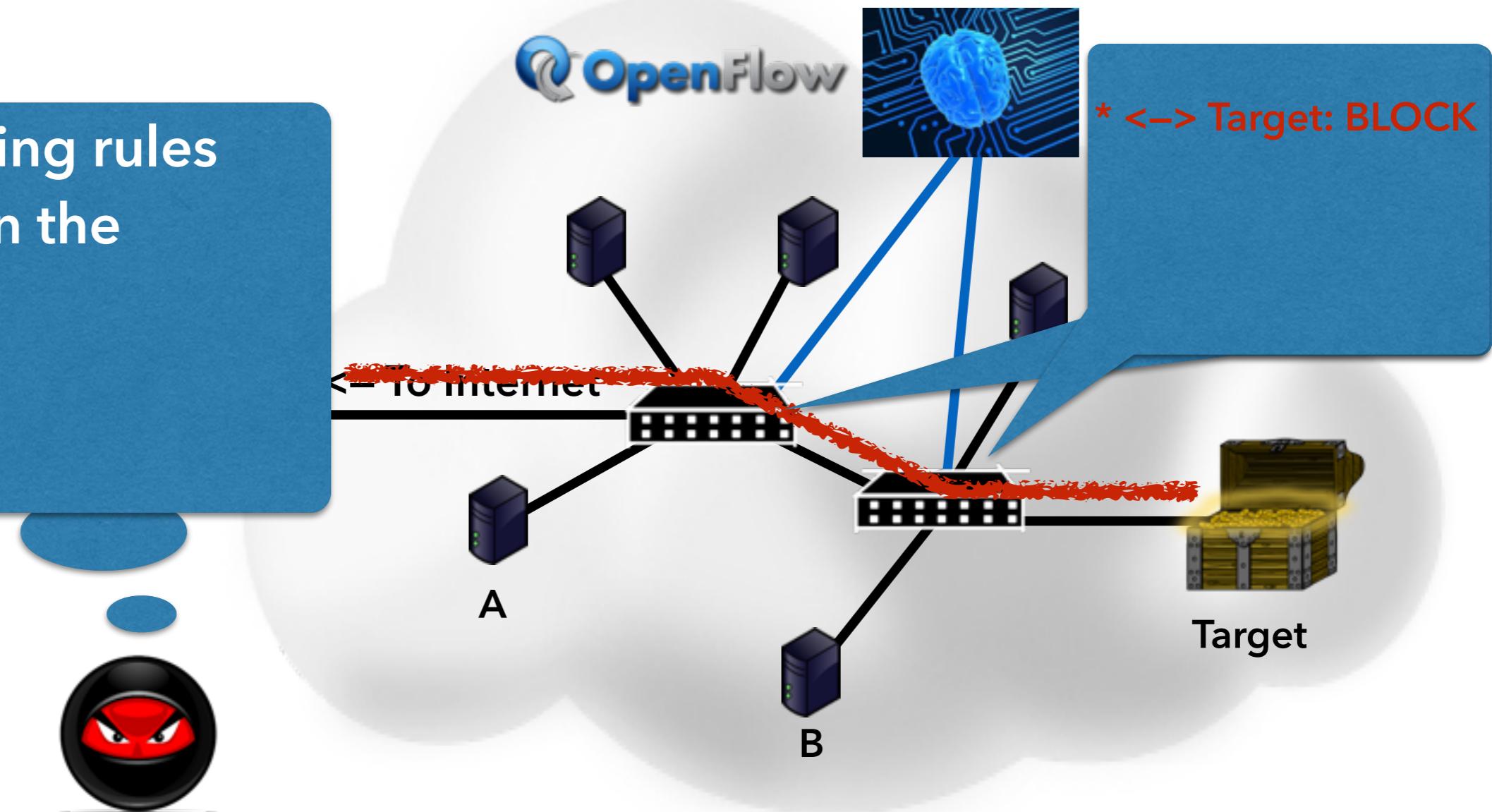
What forwarding rules
are installed on the
switches?



Motivation: Plan Multi-Staged Attacks

What forwarding rules
are installed on the
switches?

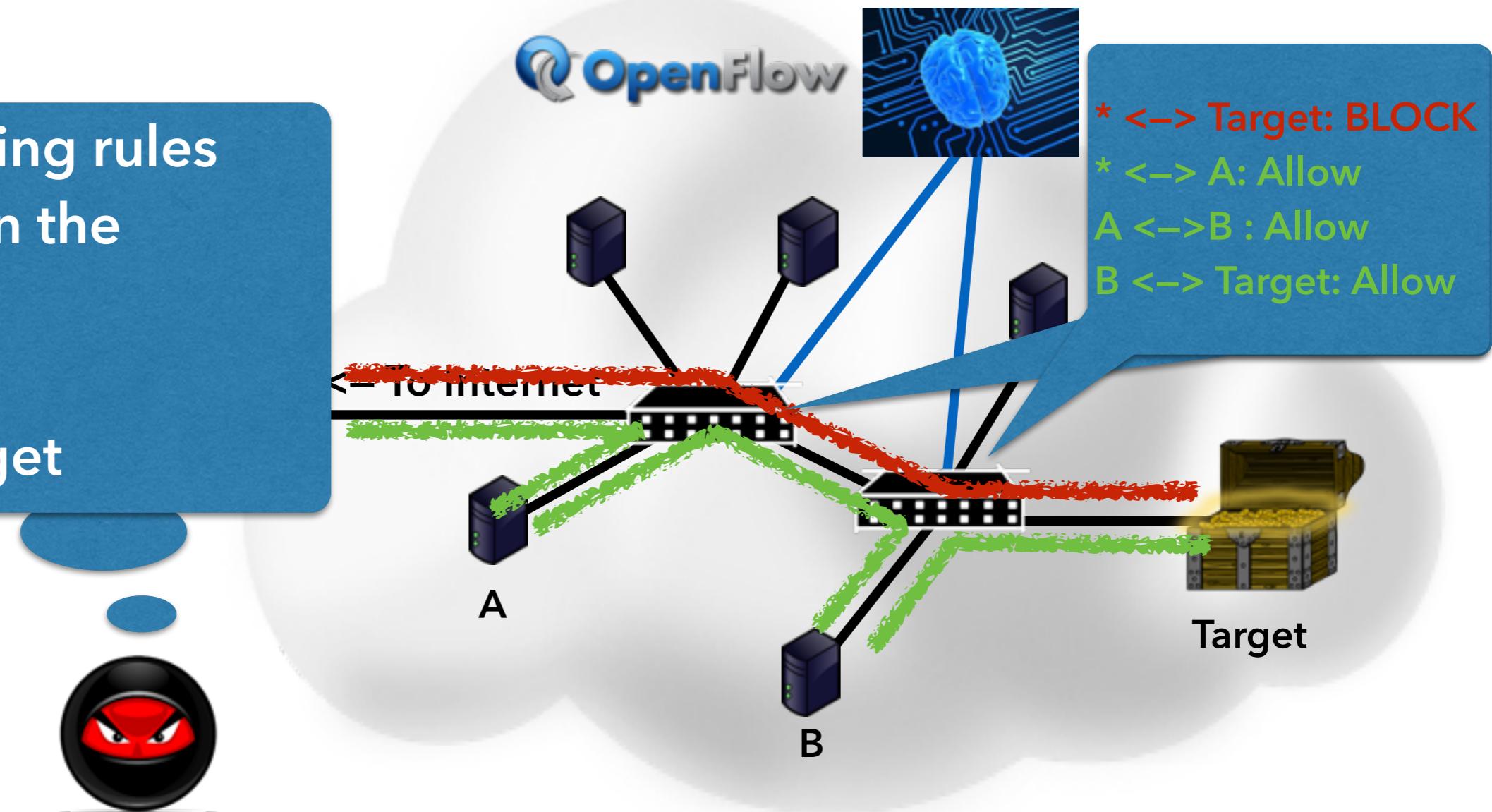
Attack Plan:
?? → Target



Motivation: Plan Multi-Staged Attacks

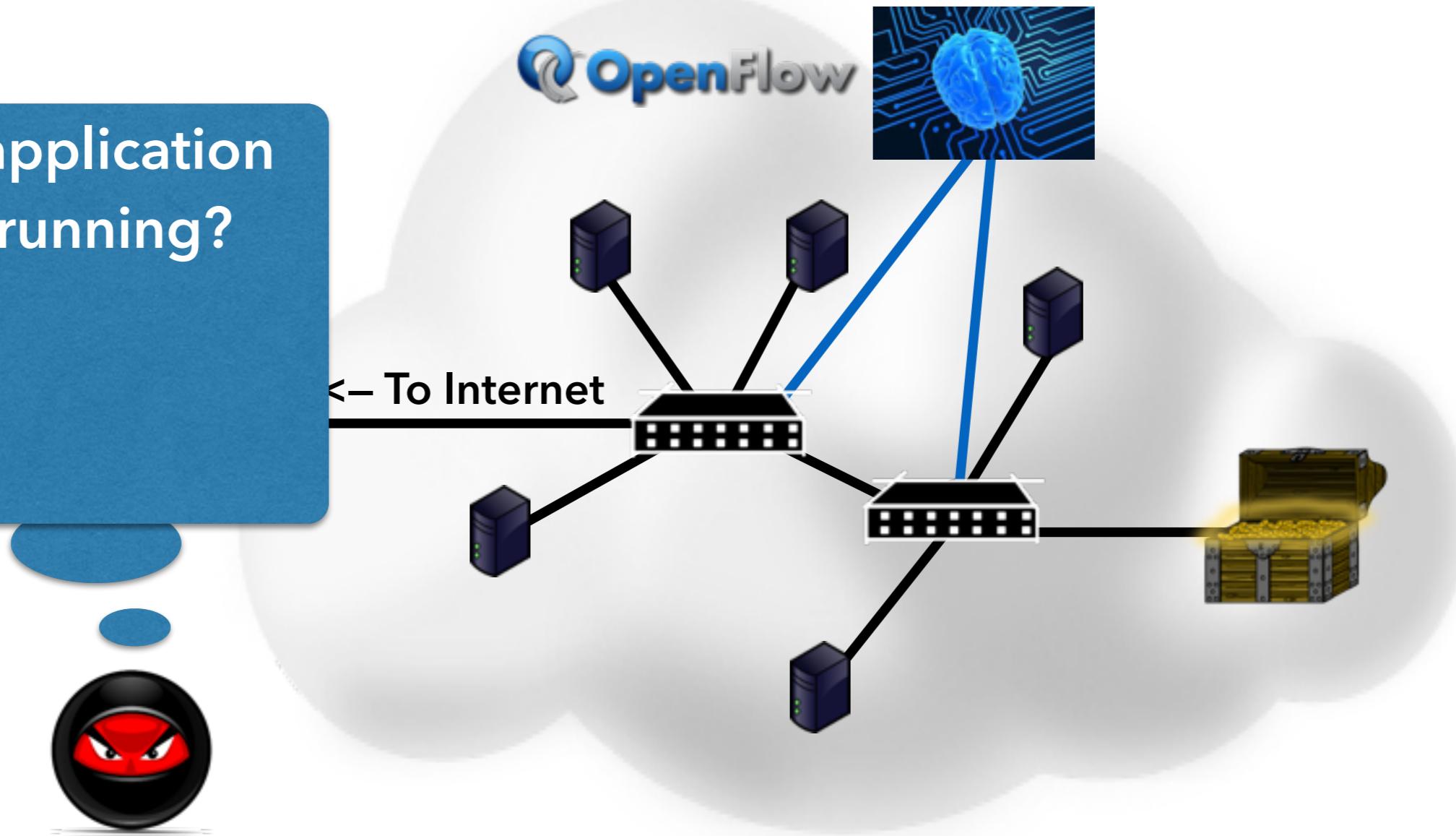
What forwarding rules are installed on the switches?

Attack Plan:
 $A \rightarrow B \rightarrow \text{Target}$



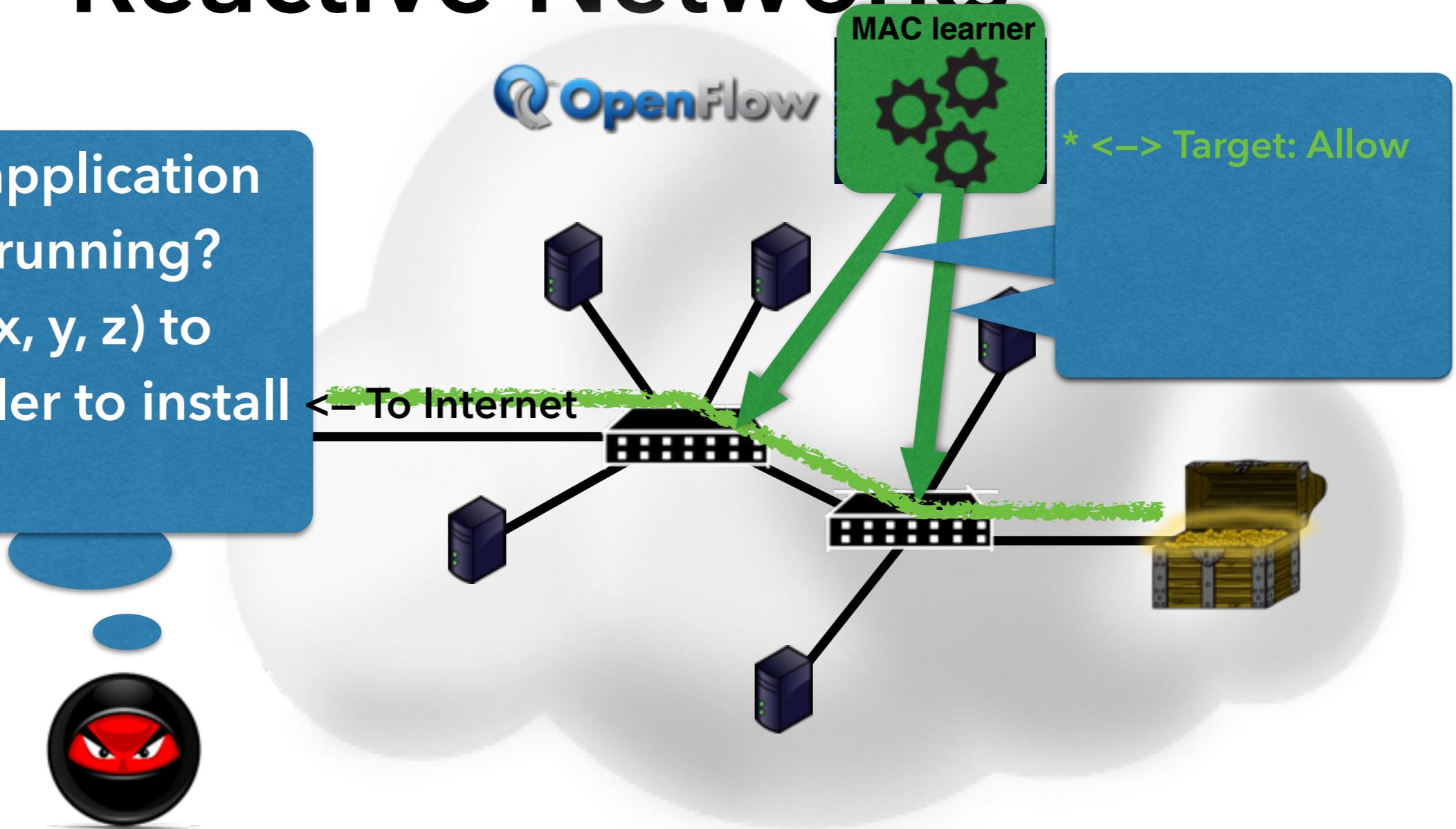
Motivation: Reprogramming Reactive Networks

What control application
is the network running?



Motivation: Reprogramming Reactive Networks

What control application is the network running?
Send packets (x, y, z) to get the controller to install this rule.



Attack Goal: Probe An OpenFlow Network to Learn its Configuration

- Which forwarding rules are installed?

Motivation: makes multi-stage attacks easier to plan

- When do new forwarding rules get installed?

Motivation: makes reactive networks easier to reprogram for adversaries

Outline

Introduction

OpenFlow Timing Attacks

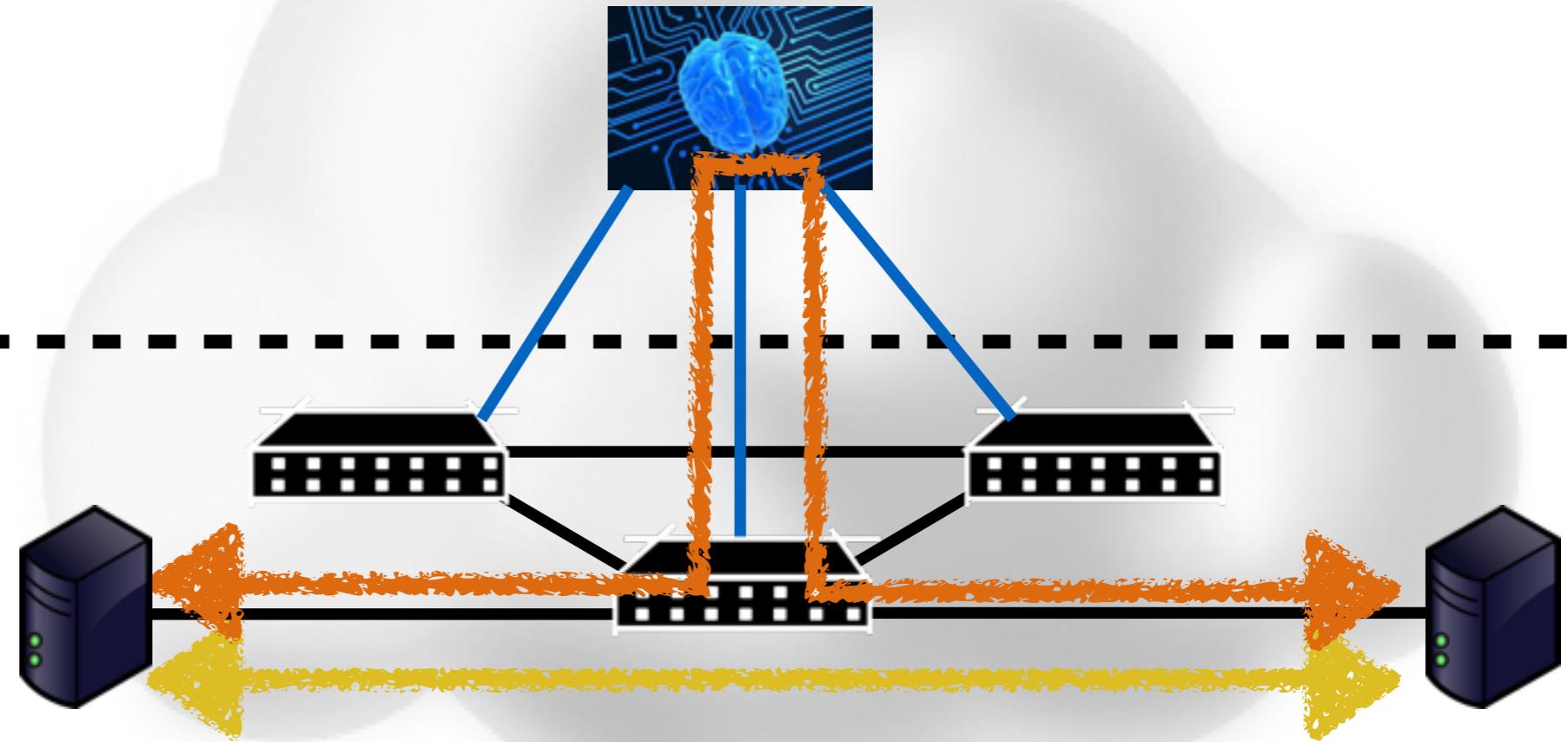
A More General OpenFlow
Timing Attack

Preliminary Results

A Simple OpenFlow Timing Property



Control Plane



Data Plane

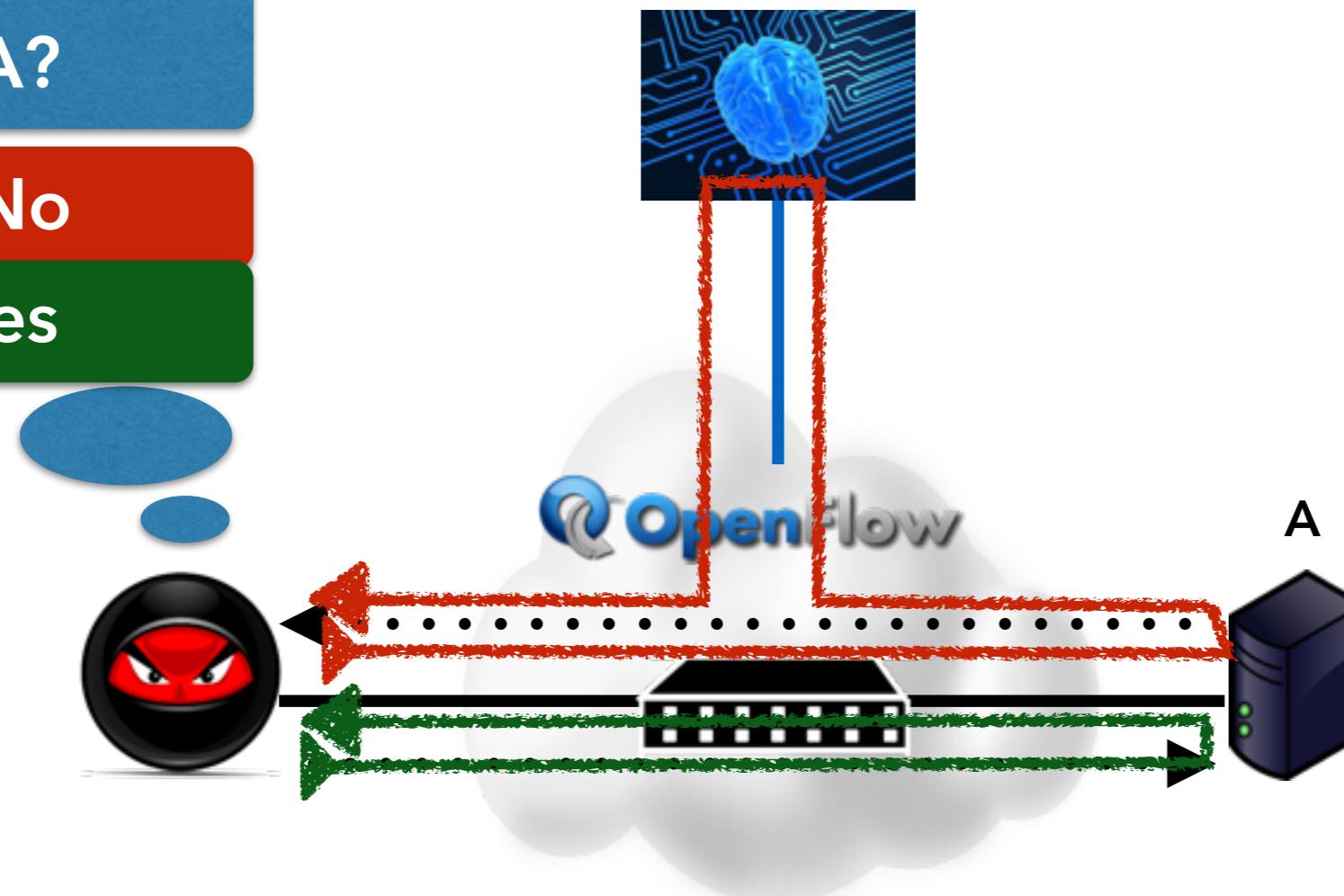
```
PING 1.1.1.2 (1.1.1.2) 56(84) bytes of data.  
64 bytes from 1.1.1.2: icmp_seq=1 ttl=64 time=2.56 ms  
64 bytes from 1.1.1.2: icmp_seq=2 ttl=64 time=0.345 ms  
64 bytes from 1.1.1.2: icmp_seq=3 ttl=64 time=0.044 ms
```

A Simple OpenFlow Timing Attack

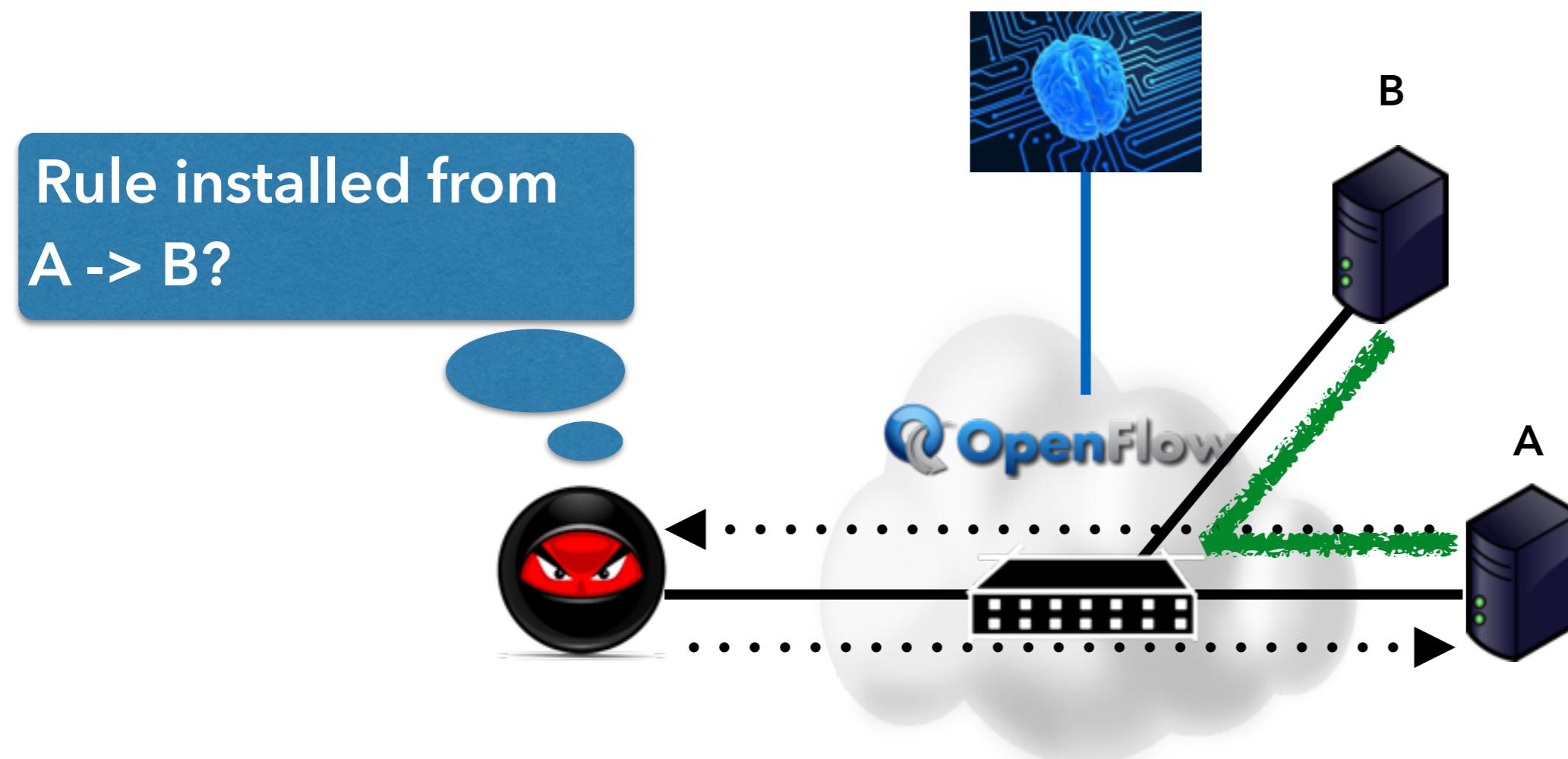
Rule installed from
My host <-> A?

High RTT → No

Low RTT → Yes

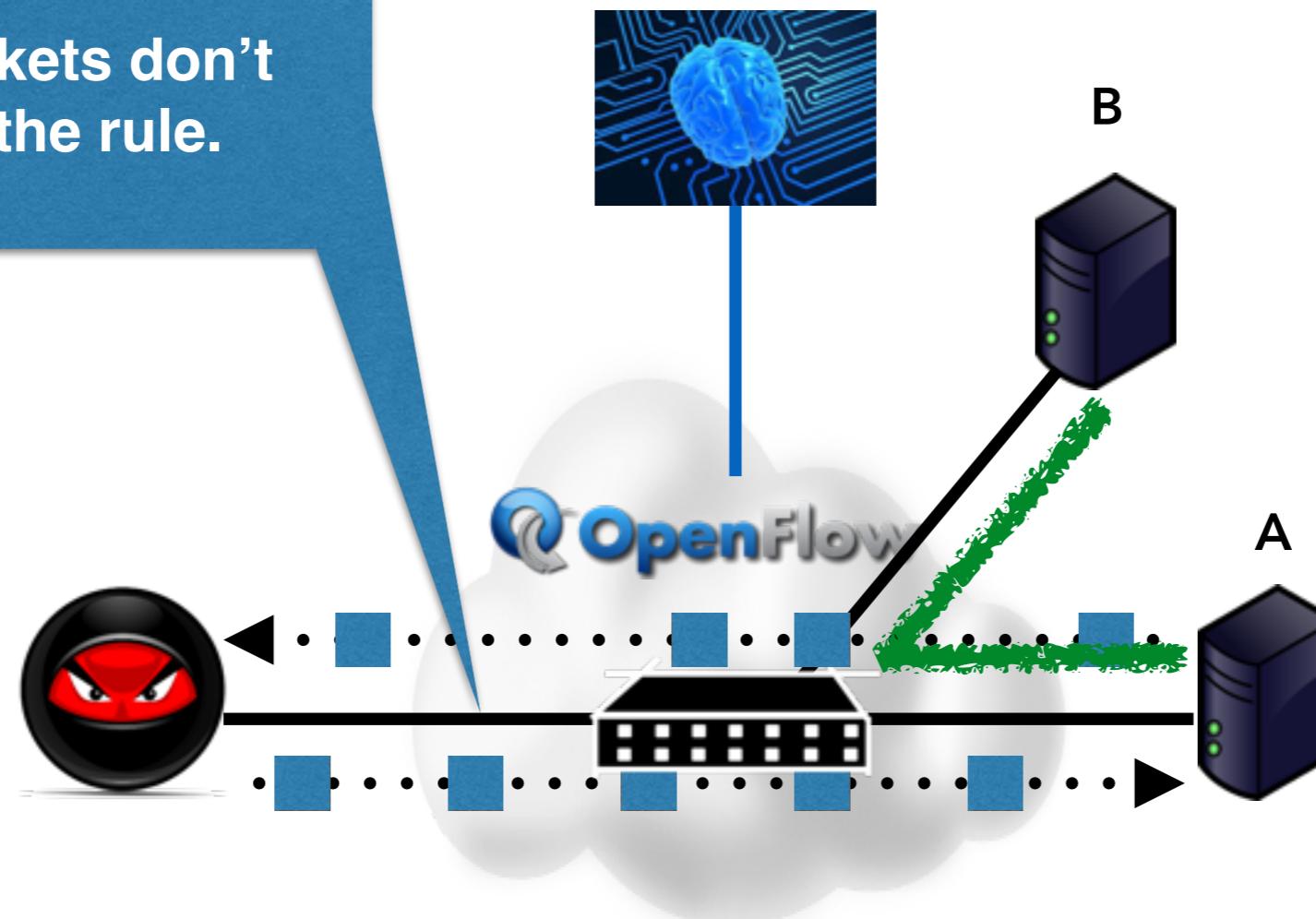


A Simple OpenFlow Timing Attack: Limitations



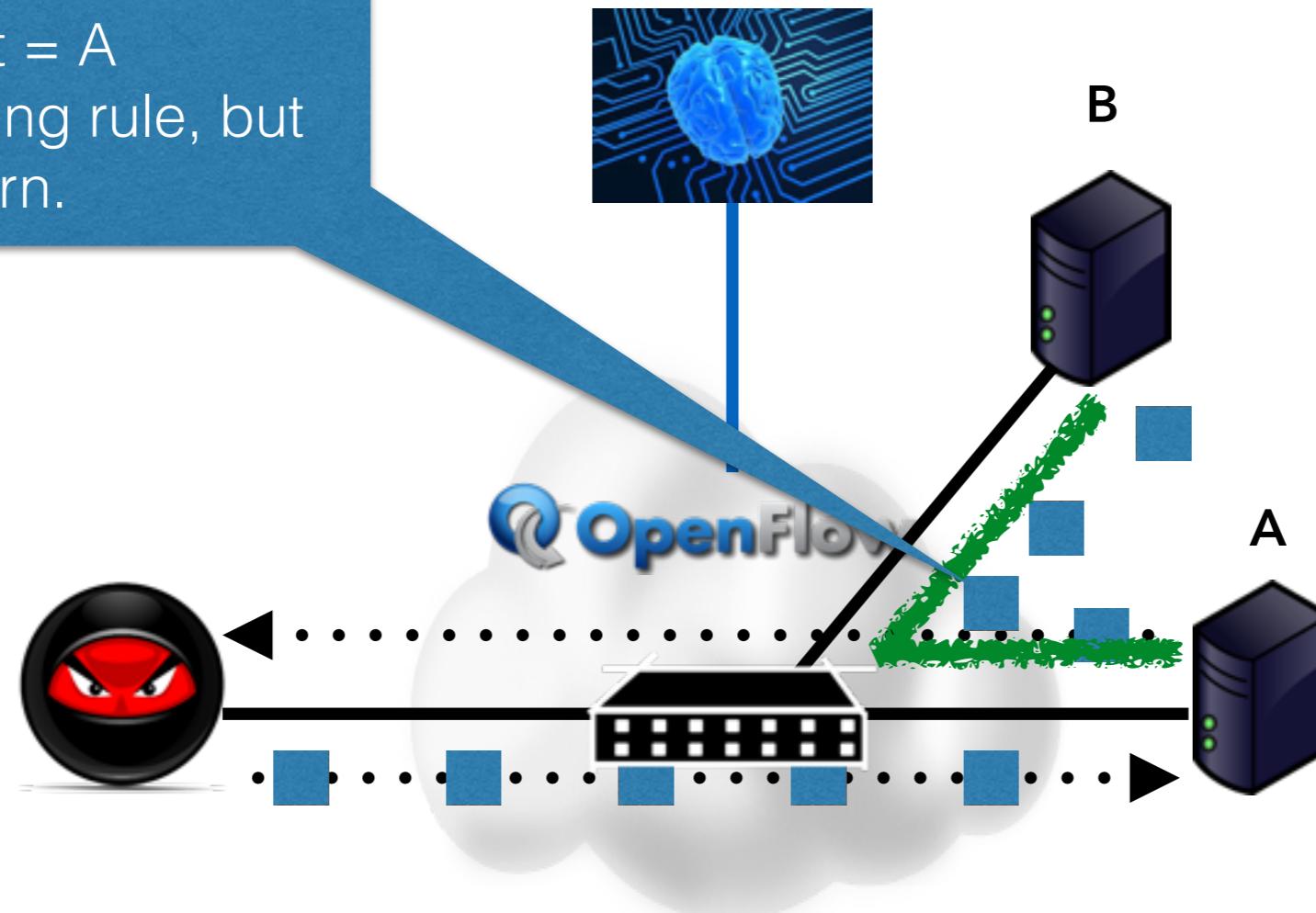
A Simple OpenFlow Timing Attack: Limitations

Ping packets don't match the rule.



A Simple OpenFlow Timing Attack: Limitations

replies to spoofed packets with
 $\text{src} = \text{B}$, $\text{dst} = \text{A}$
match the forwarding rule, but
don't return.



Previous OpenFlow Timing Attacks Based on this Property

Do switches have aggregate rules installed? [1]
How large are switch flow tables? [2]



[2] J. Leng, Y. Zhou, J. Zhang,

Can we design a more general attack to learn which forwarding rules are installed on switches, and when they get installed?

[1]

Outline

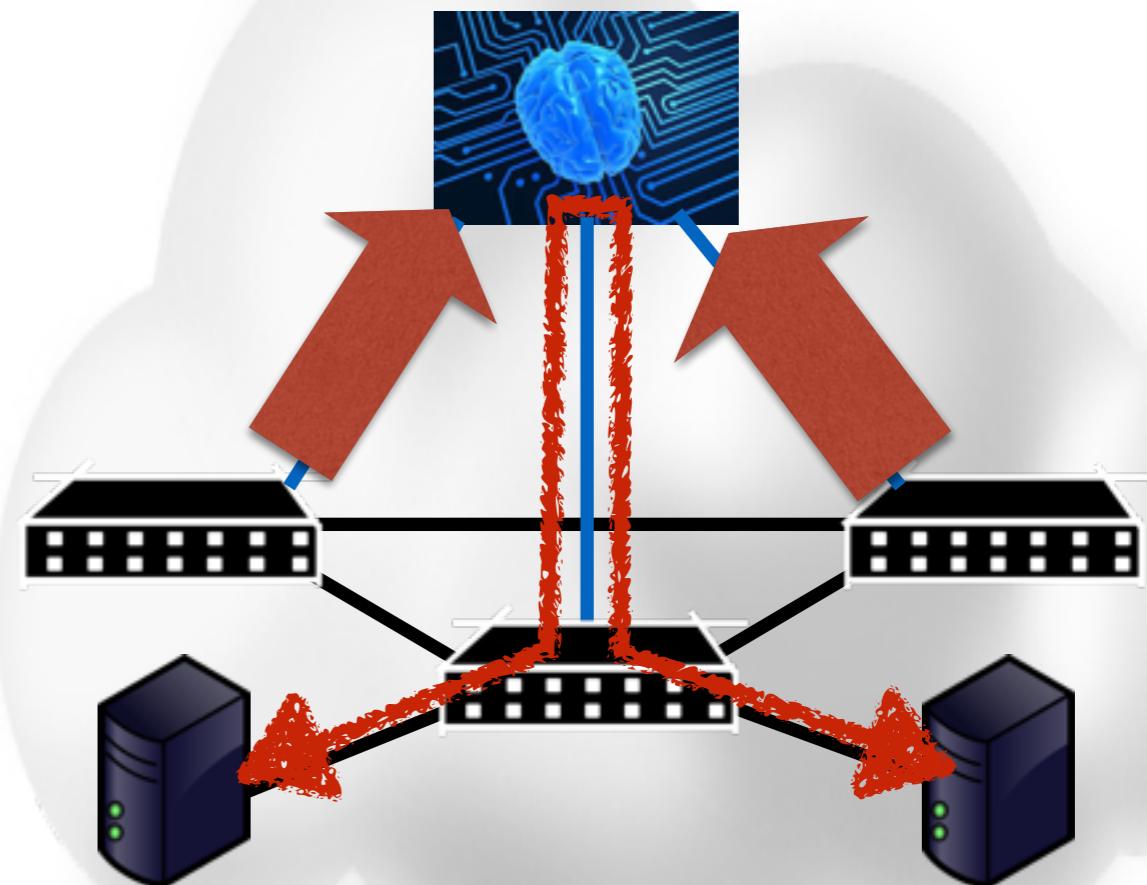
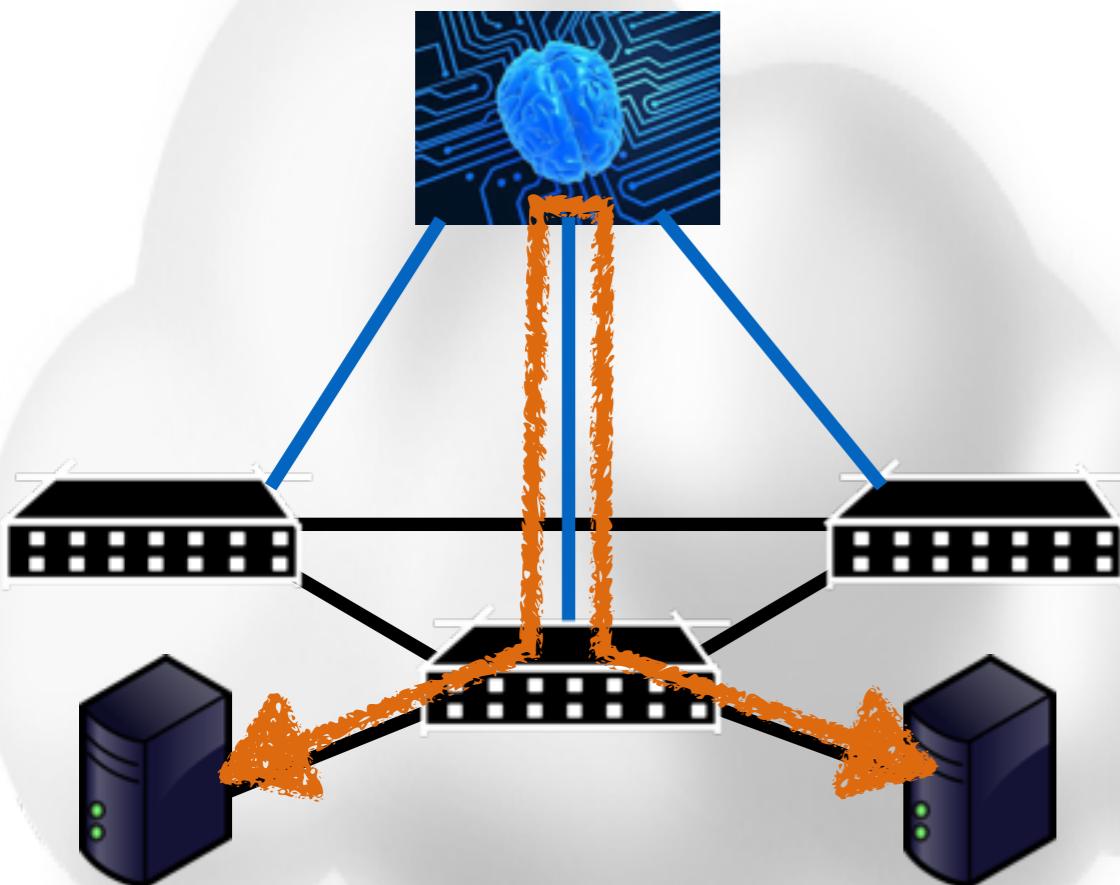
Introduction

OpenFlow Timing Attacks

A More General OpenFlow
Timing Attack

Preliminary Results

OpenFlow Timing Property: Control Planes Response Time Depends on Load

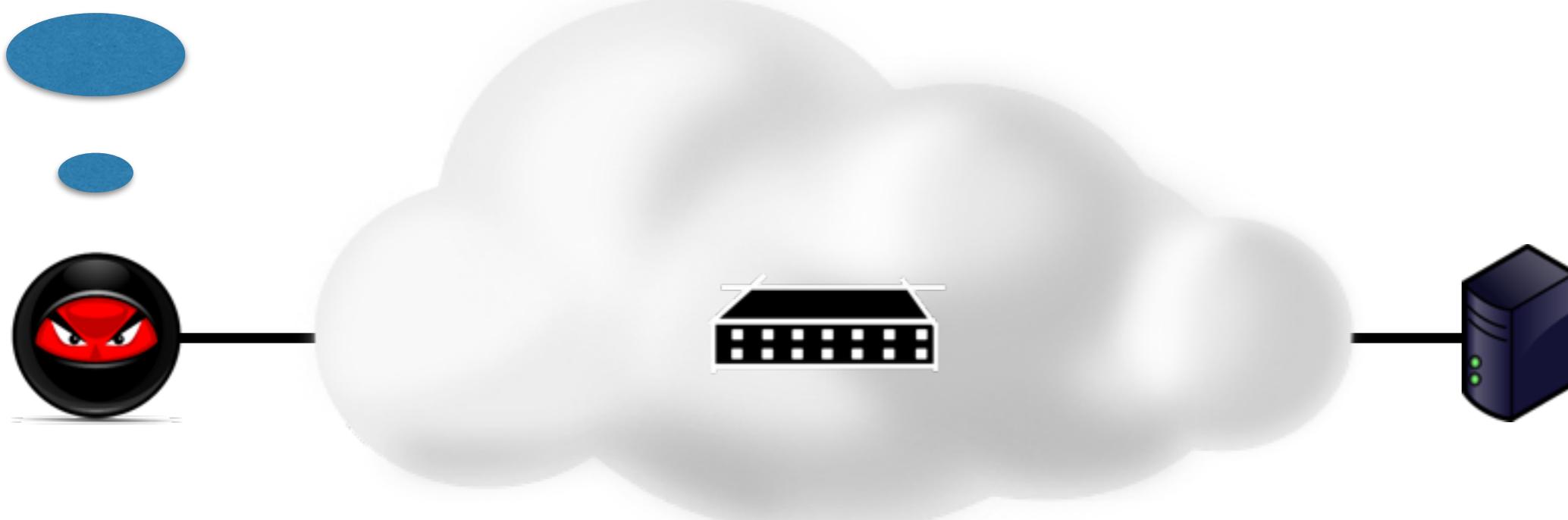


```
PING 1.1.1.2  
64 bytes from 1.1.1.2: time=2.56 ms  
64 bytes from 1.1.1.2: time=0.345 ms  
64 bytes from 1.1.1.2: time=0.044 ms
```

```
PING 1.1.1.2  
64 bytes from 1.1.1.2: time=10.8 ms  
64 bytes from 1.1.1.2: time=0.345 ms  
64 bytes from 1.1.1.2: time=0.044 ms
```

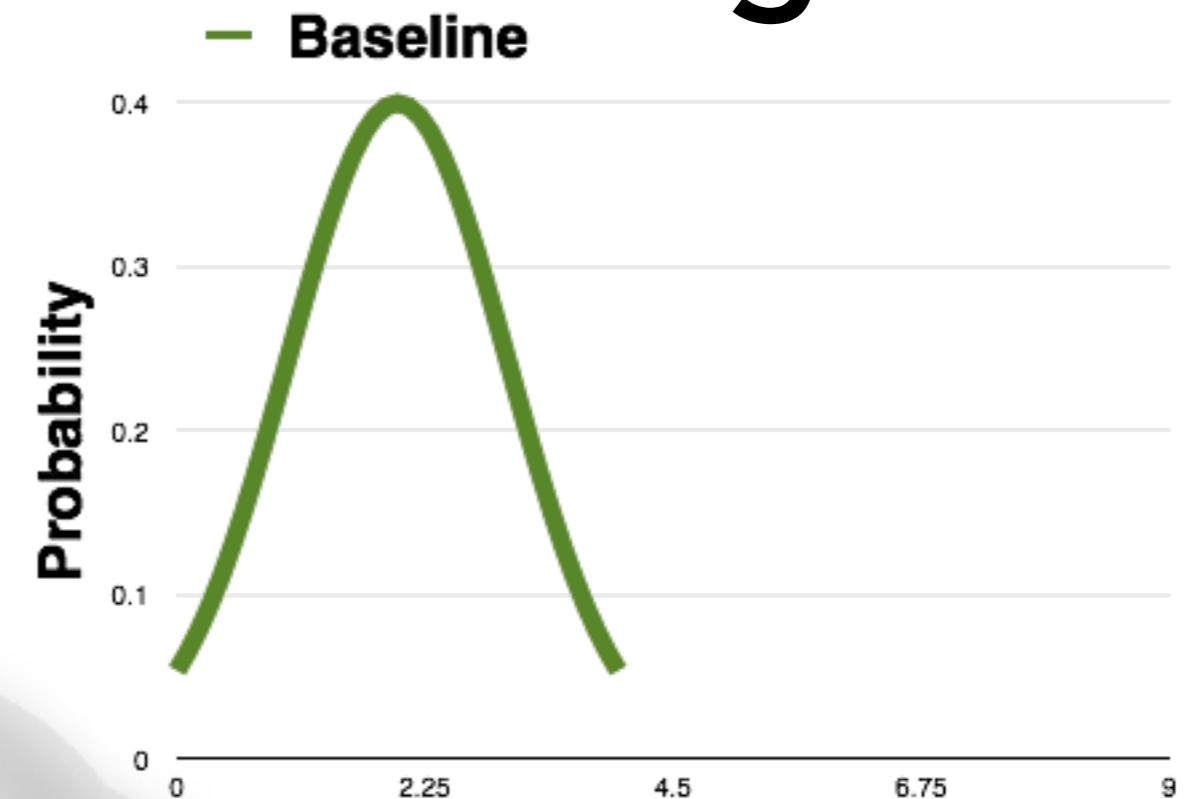
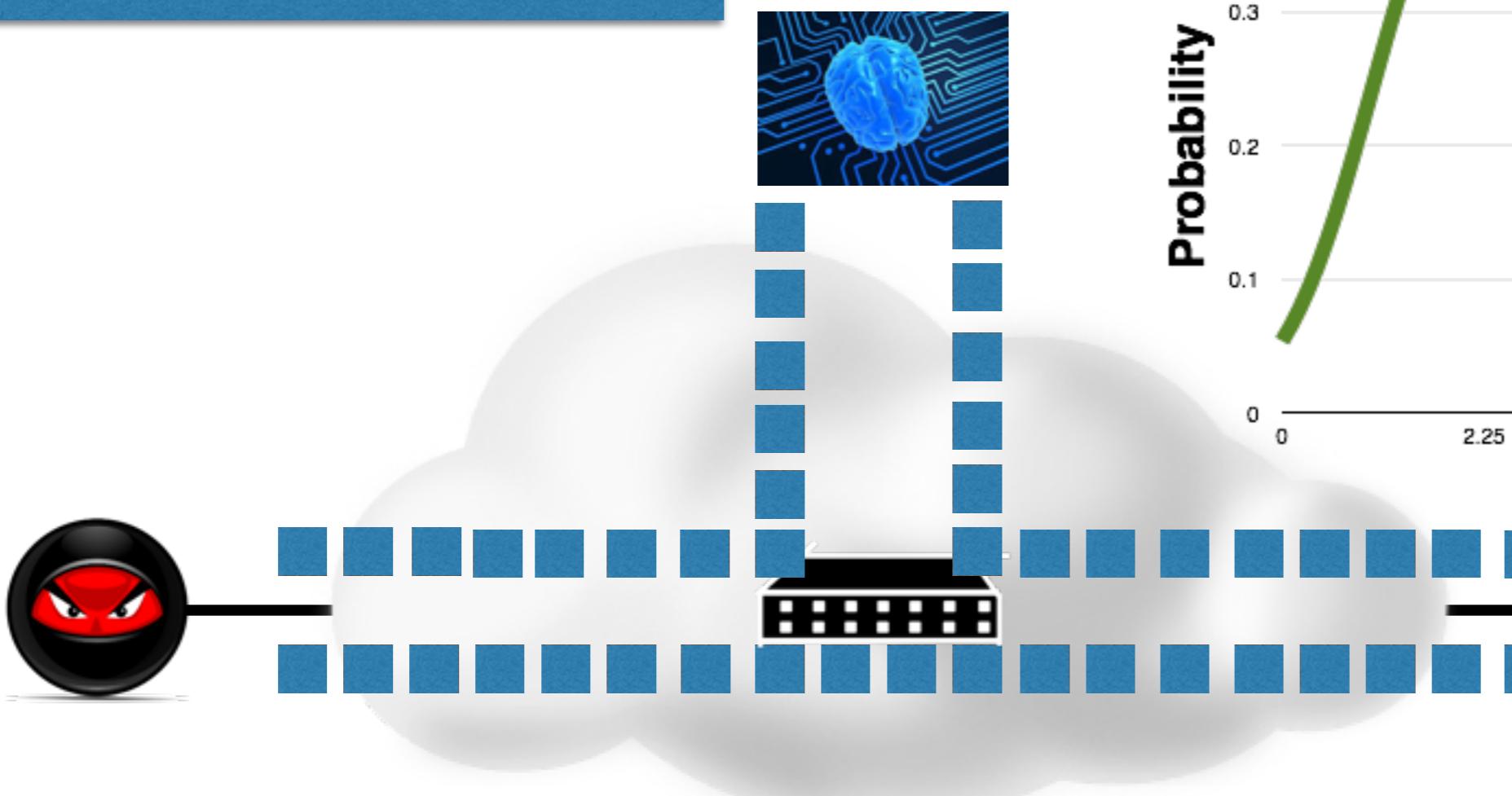
The Basic Approach: Estimating Control Plane Load Change

Will these packets
match a rule?



The Basic Approach: Estimating Control Plane Load Change

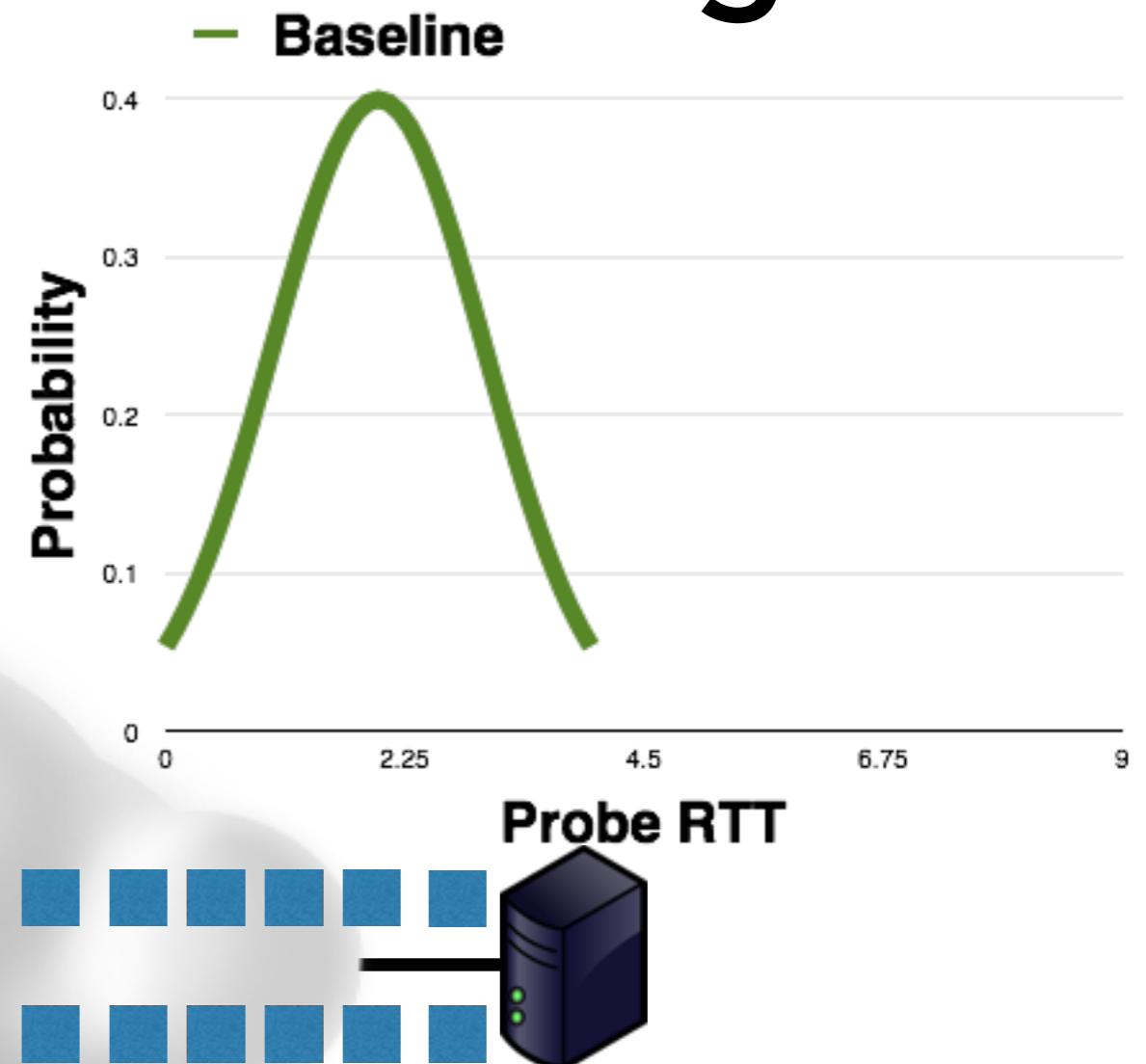
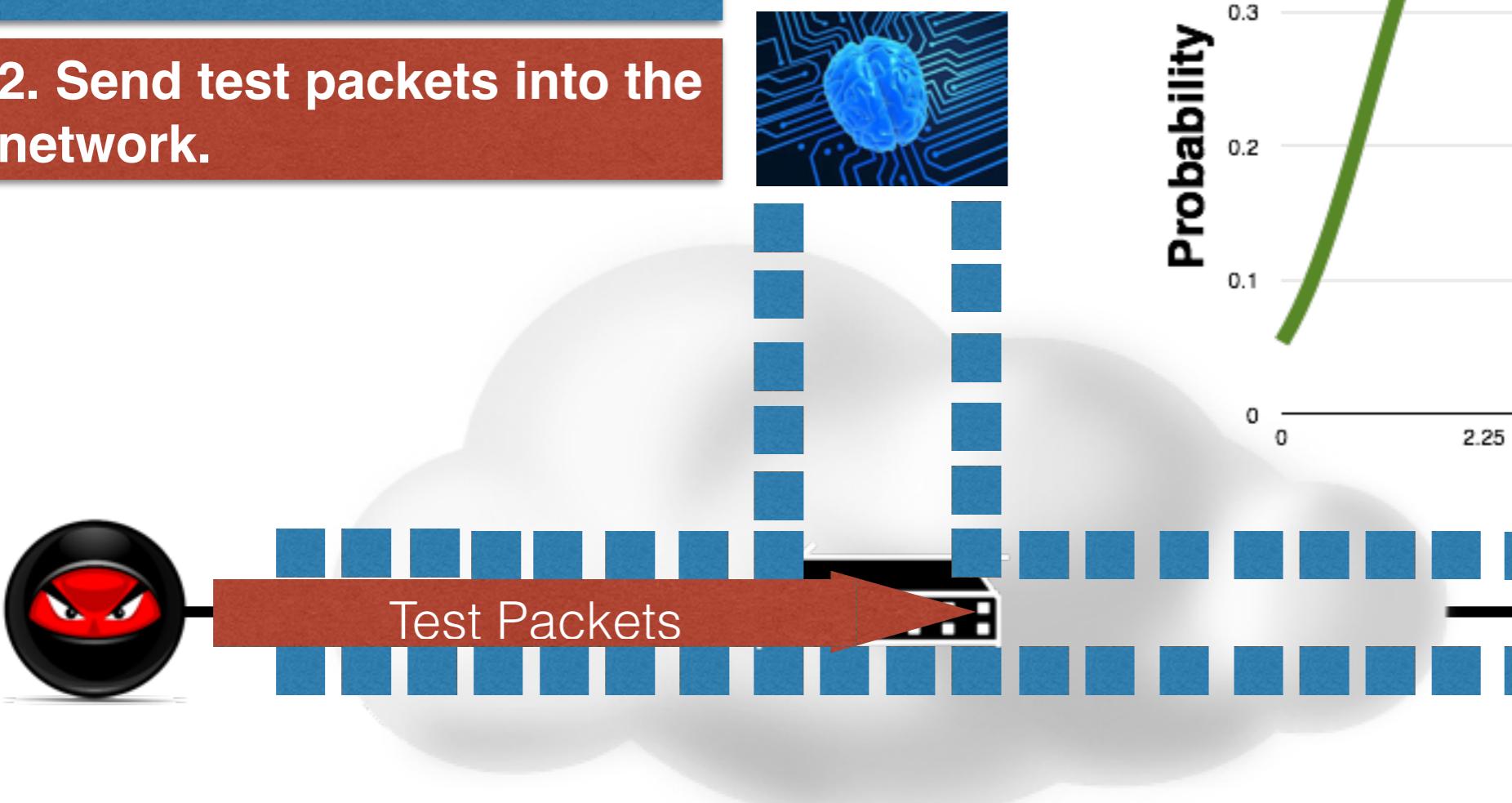
1. Send timing probes through the control plane, measure RTT.



The Basic Approach: Estimating Control Plane Load Change

1. Send timing probes through the control plane, measure RTT.

2. Send test packets into the network.

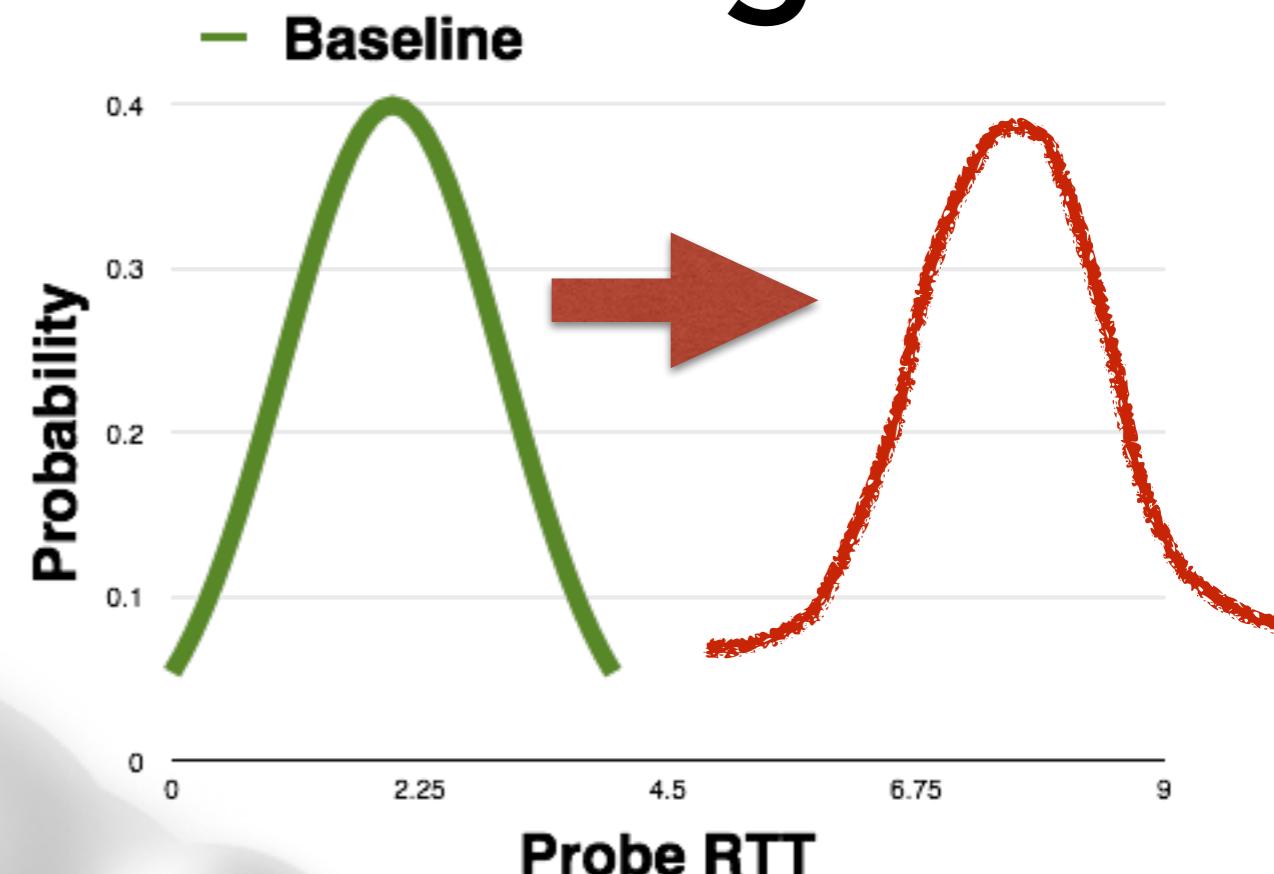
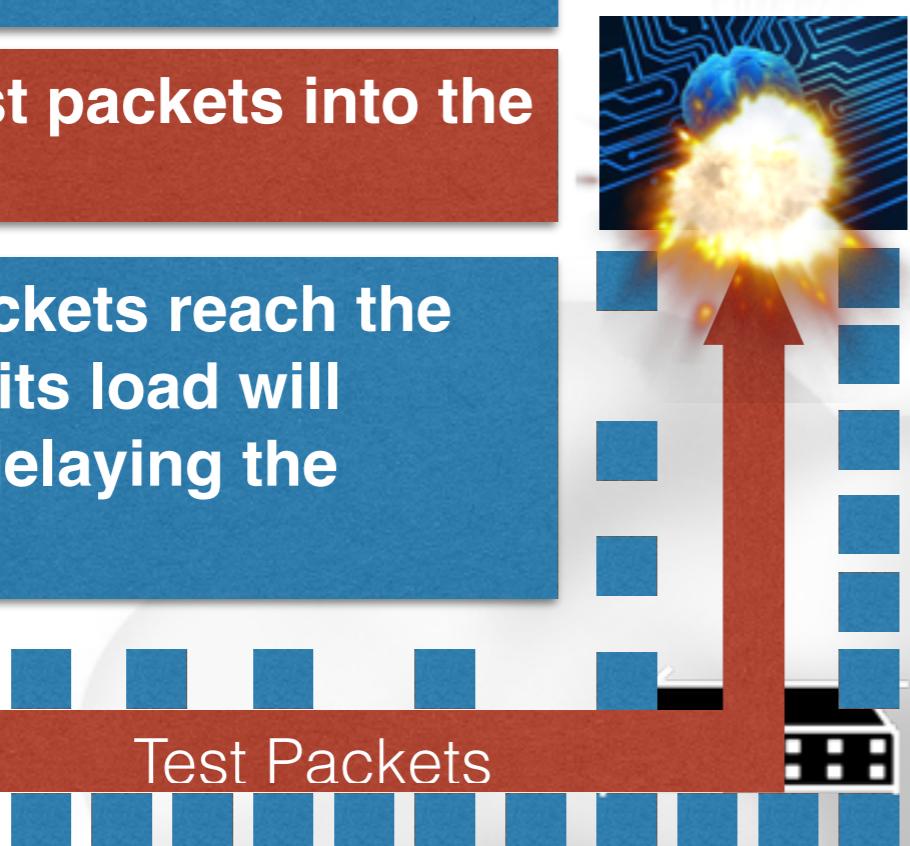


The Basic Approach: Estimating Control Plane Load Change

1. Send timing probes through the control plane, measure RTT.

2. Send test packets into the network.

3. If the packets reach the controller, its load will increase, delaying the probes.

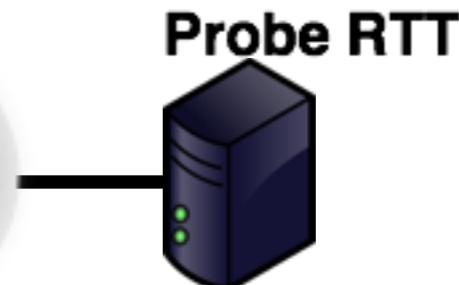
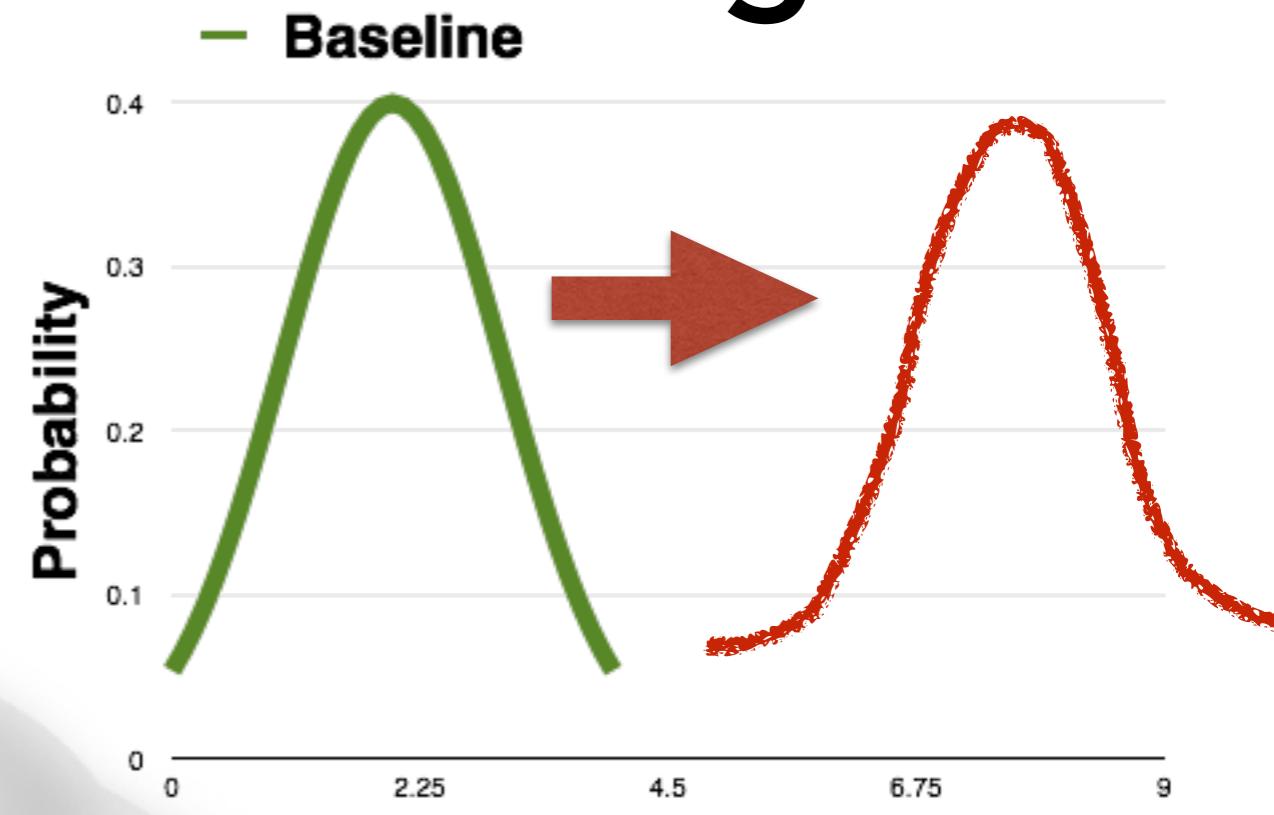
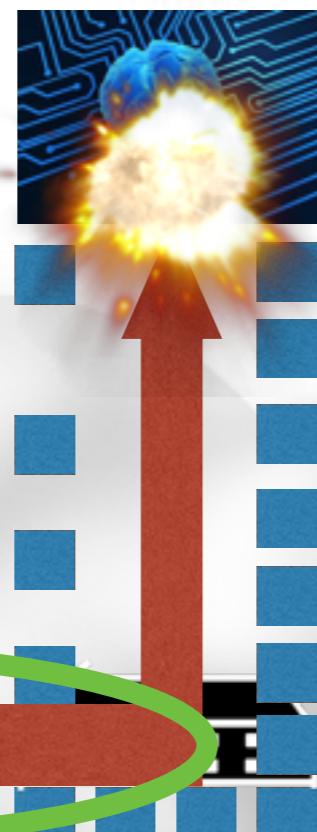


The Basic Approach: Estimating Control Plane Load Change

1. Send timing probes through the control plane, measure RTT.

2. Send test packets into the network.

3. If the packets reach the controller, its load will increase, delaying the probes.



Learning Higher Level Configuration Details

Which forwarding rules are installed in the switches?

Which fields are wildcarded?

What kind of application is the controller running?

Which packets reach the controller?

Which sequences of packets cause the controller to install flows?



Outline

Introduction

OpenFlow Timing Attacks

A More General OpenFlow
Timing Attack

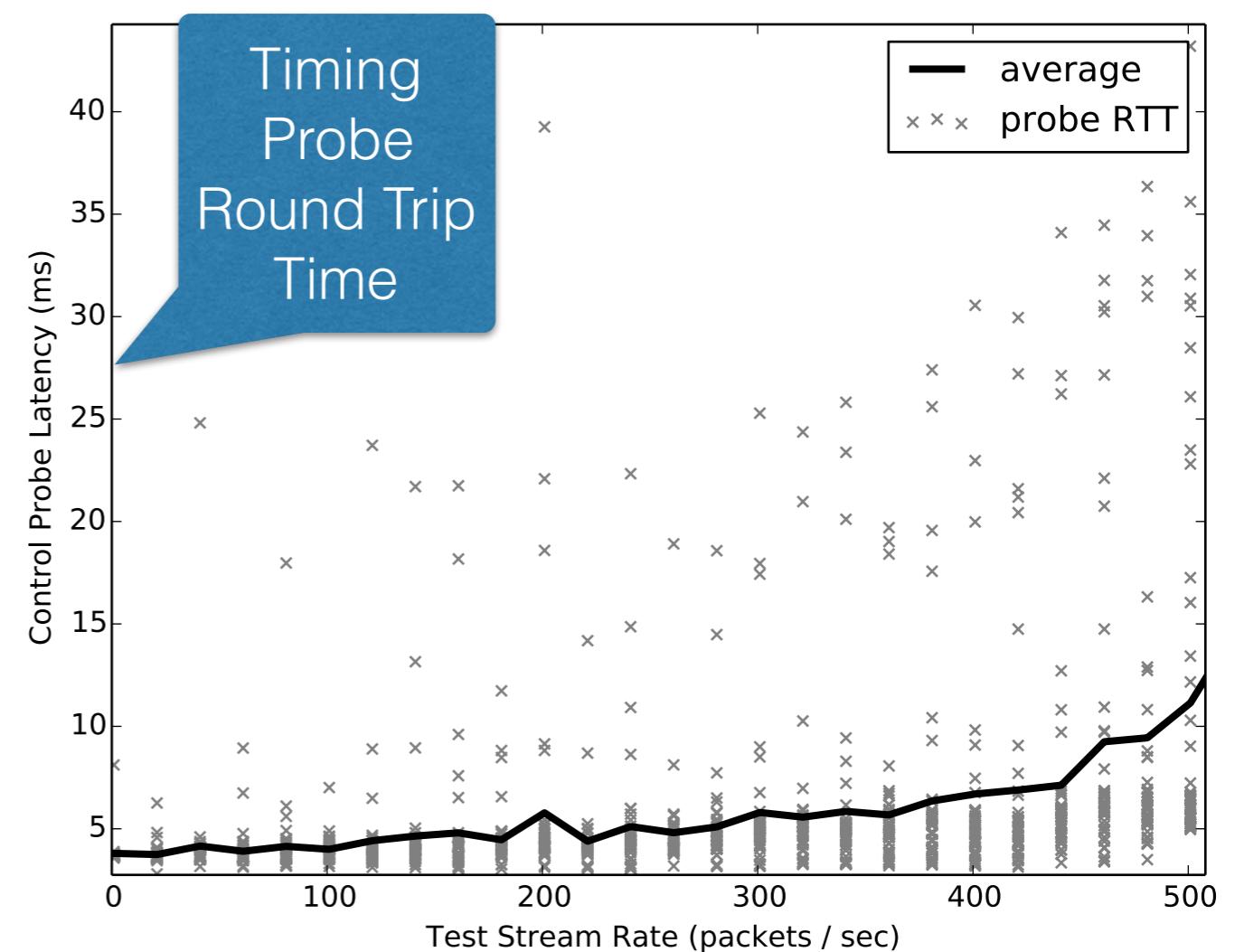
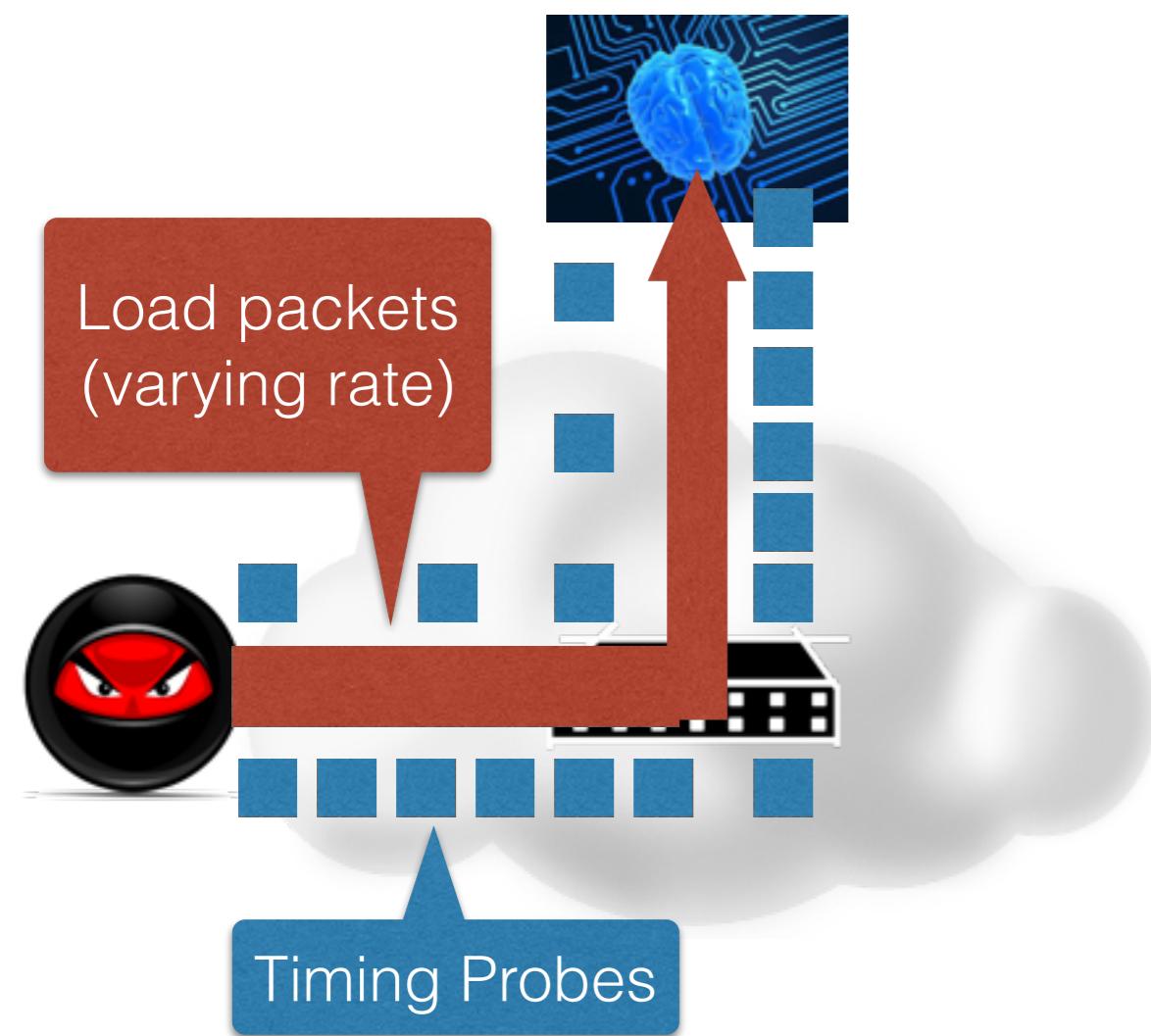
Preliminary Results

Preliminary Results

Does probe RTT estimate controller load?

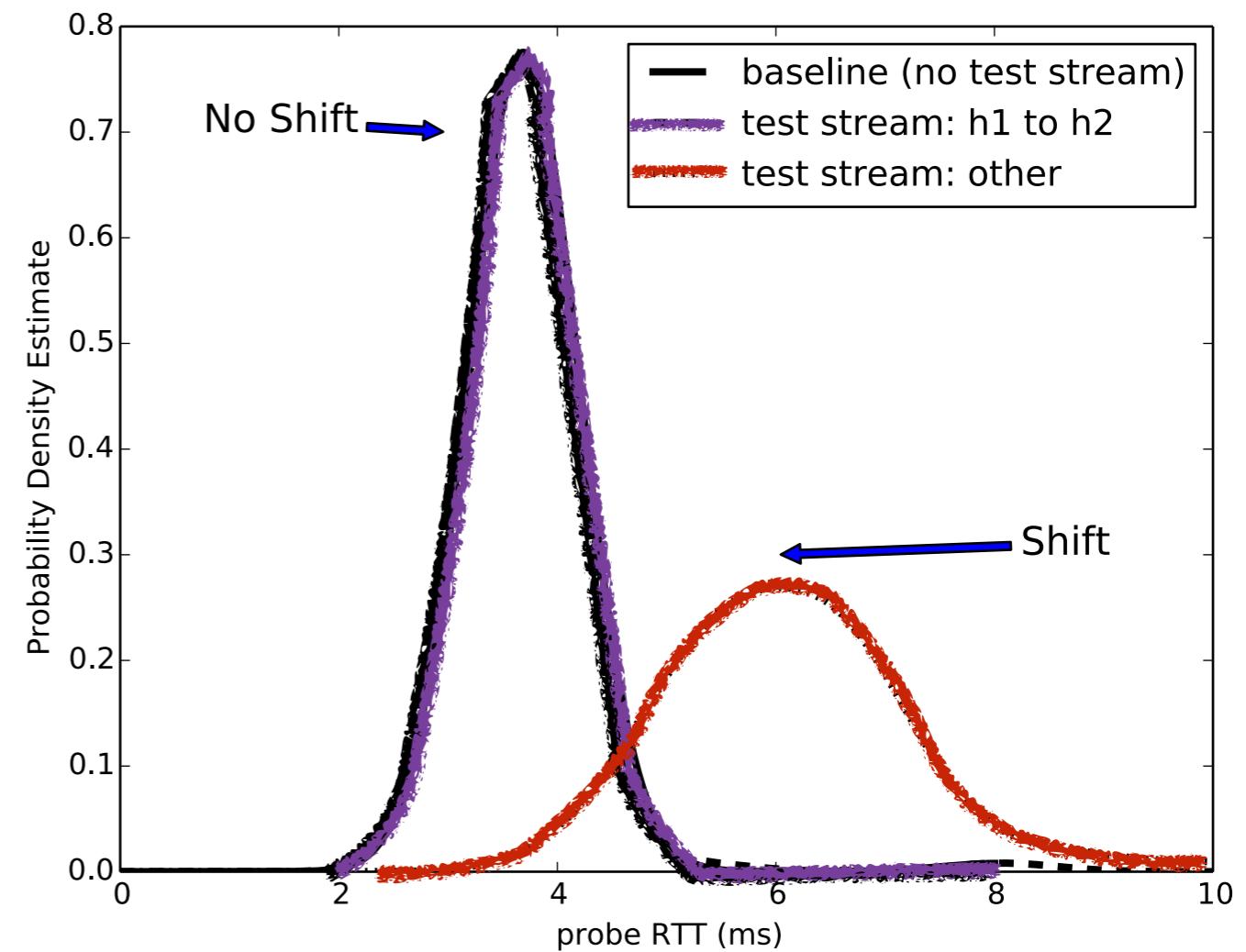
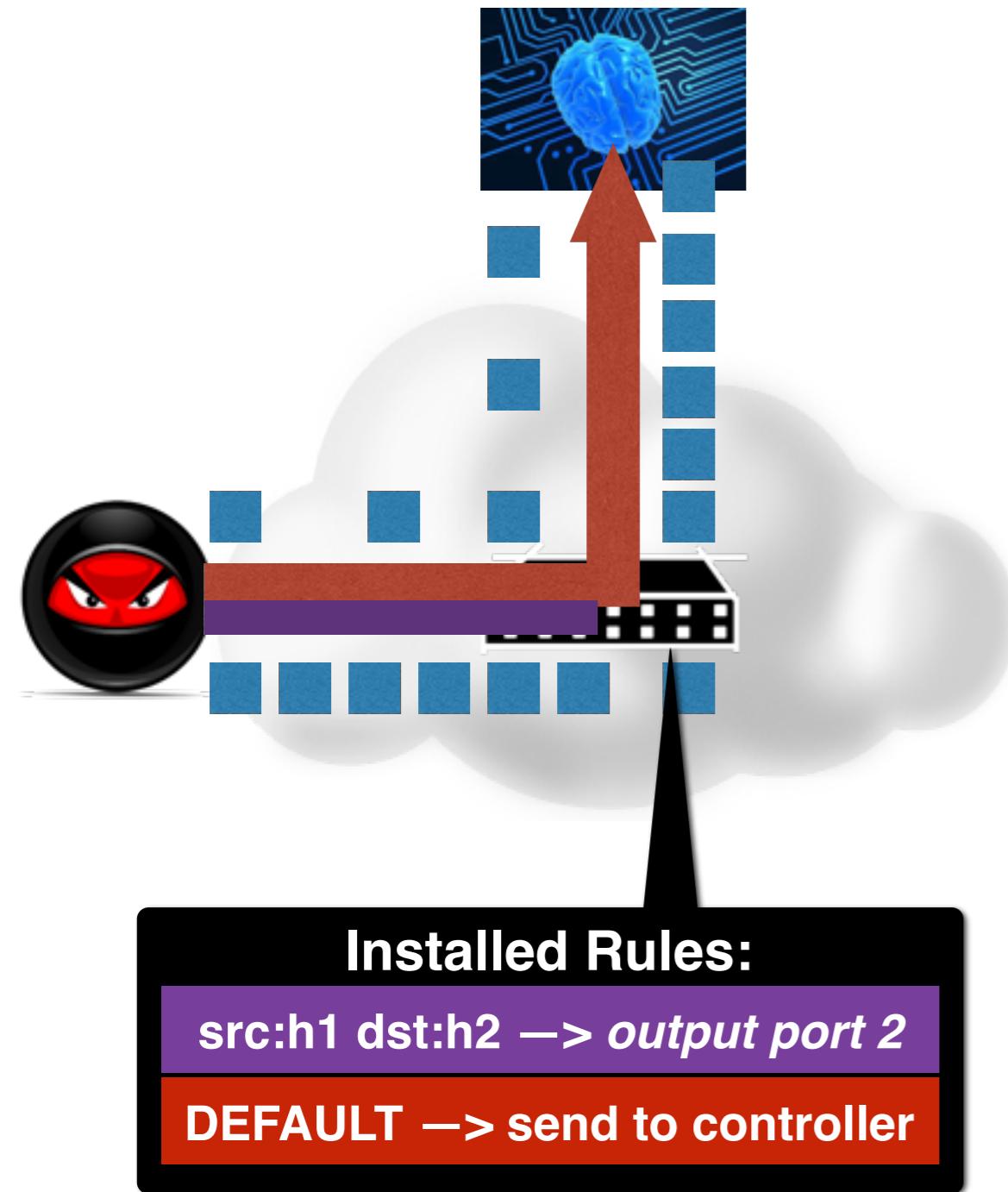
Can an adversary learn if a forwarding rule is installed?

Estimating Controller Load



Packets per second sent to controller

Learning if a Rule is Installed



More in the Paper

Timing SDN Control Planes to Infer Network Configurations

ABSTRACT

In this paper, we study information leakage by control planes of software-defined networking (SDN) providers to evaluate the security of their network configurations. We test the inference capability of an OpenFlow control plane deployed in our testbed and we discuss an inference attack that an adversary with control of a single host could use to learn about network configurations without needing to compromise any network infrastructure (i.e., switches or controller servers). We also demonstrate that our inference attack works on real OpenFlow hardware. To our knowledge, no previous work has evaluated inference attacks outside of simulation.

1. INTRODUCTION

Software-defined networking (SDN) promises to revolutionize the way we manage networks by separating the interface to network elements from the programmable logic, and organizations such as Facebook, Google, and the NSA have already deployed large-scale SDN networks [1]. A key benefit of SDN is the separation of the control and data planes in a network. The data plane forwards data across the network at line rate by matching packets against simple forwarding rules. While the control plane manages the network configuration, it can also perform advanced packet processing or modify a packet's header fields (e.g., when a packet contains headers for forward a packet).

Processing a packet in the control plane takes orders of magnitude more time than processing a packet in the data plane. These timing differences tell information about a network. Previous work showed that by measuring the response times of servers in a DCN, an adversary can perform a man-in-the-middle attack that has just passed access to it in a network [2]. In this paper, we show how an adversary can learn about the rules in the control plane by leveraging timing differences and whether a controller links that appear in flows [3].

In this paper, we study a more sophisticated inference attack that an adversary can use to infer much more about a network. Our core observation is that how long the control plane takes to process a packet is related to the time it takes to process the test streams. An adversary can time the test streams with enough precision (e.g., up to 50%) to determine whether the control plane is reaching the control plane or causing the control plane to install new forwarding rules even if the packets do not match a rule from hosts in the DCN. To continue, previous SDN inference attacks could only determine whether legitimate requests to servers in the DCN were through the control plane. The main difference here is that our attack shows an adversary who controls a single host in the DCN can learn about the rules in a network's forwarding table and the controller's logic. An adversary can use this knowledge to better plan subsequent stages of attacks. For example, we show how an adversary can learn which sequences of packets cause the controller to install forwarding rules. Once an adversary learns this, they can exert switch forwarding tables and greatly degrade the entire network's performance by only writing a relatively small number



Figure 1: Summary of the control plane inference attack: an adversary sends probes that travel through the control plane while sending test packets into the network. If the control plane processes the test packets the probe RTT will increase.

Test Packet Stream:
 $\text{test_packet_stream} ::= \langle \text{tempfile}, \text{size}, \text{maxcount}, \text{rate} \rangle$

Stream Template:
 $\text{template} ::= \langle \text{header_field} = \text{field_value}, \dots \rangle$

Header Fields:
 $\text{header_field} ::= \text{mac_source} \sqcup \text{mac_dest} \sqcup (\text{ip_source} \sqcup \text{ip_dest}) \sqcup \dots$

Header Values:
 $\text{field_value} ::= \begin{cases} \text{c} & \text{I.e., each packet in the stream will have the same constant value c for this field} \\ \text{r} & \text{I.e., each packet in the stream has a random value for this header} \\ \text{c}_i & \text{The header field value for the i-th packet in the stream will be C}_i \end{cases}$

Figure 2: Syntax for test packet streams.

test before.

OpenFlow Echo Messages If the OpenFlow controller is reachable from the data plane (i.e., there is no isolated control network or VLAN), the adversary can simply send an OpenFlow Echo message to the controller. The controller will reply with an Echo response containing the original Echo message. The RTTs of echo request/response pairs can be used to time the control plane.

Spoofer ARP Requests In an OpenFlow network, MAC learning is controlled by the controller. The OpenFlow switch sends the first packet from a new MAC address to the controller, so that the controller can figure out which rules to install on the switch to forward future packets to the right host. If the adversary sends an ARP request to another host on the network using a spoofed source MAC address, the request and/or reply will be routed through the control plane, and the RTTs of ARP request/reply pairs can be used to time the control plane. We use this technique in the evaluation in Section 4.

2. INFERENCE ATTACK OVERVIEW

Our threat model is based on an adversary that has root access to a single host in a network and wishes to learn as much information as possible about how the network operates without needing to compromise any other devices in the network, including the switch and controller. This model reflects two common use cases: an adversary performing a man-in-the-middle attack that has just passed access to it in a network or launching an attack on a target organization and need to learn about the target's forwarding table structure and whether a controller links that appear in flows [3].

In this paper, we study a more sophisticated inference attack that an adversary can use to infer much more about a network. Our core observation is that how long the control plane takes to process a packet is related to the time it takes to process the test streams. An adversary can time the test streams with enough precision (e.g., up to 50%) to determine whether the control plane is reaching the control plane or causing the control plane to install new forwarding rules even if the packets do not match a rule from hosts in the DCN. To continue, previous SDN inference attacks could only determine whether legitimate requests to servers in the DCN were through the control plane. The main difference here is that our attack shows an adversary who controls a single host in the DCN can learn about the rules in a network's forwarding table and the controller's logic. An adversary can use this knowledge to better plan subsequent stages of attacks. For example, we show how an adversary can learn which sequences of packets cause the controller to install forwarding rules. Once an adversary learns this, they can exert switch forwarding tables and greatly degrade the entire network's performance by only writing a relatively small number

of packets. In this paper, we explain the inference attack in greater detail, and we demonstrate its feasibility on a testbed with real OpenFlow switches and controllers.

3. EVALUATION

The adversary controls host h_1 , and could send arbitrary test packets into the network. If the packets in a test stream end up being processed by the control plane, the RTT of the adversary's probes will increase. By setting one or more packet header fields to the same value for all packets in a stream, the adversary can learn the choice of config rule installed in the control plane.

For example, if an adversary wanted to know whether there were destination-based forwarding rules installed in the data plane for a particular MAC address, they could send a test packet stream with a fixed MAC destination address and random MAC source addresses. If the RTT of the adversary's timing probes increases while sending the test streams, then the controller must be processing the test streams, implying that no rule is installed. If the RTT of the timing probes does not increase, the controller installs a matching forwarding rule onto the switch that forwards off packets with that MAC address out of the associated port. Otherwise, the controller installs the switch to forward the packet.

3.1 Timing the Control Plane

To perform an inference attack, an adversary must have some way of estimating how control-plane timing changes with respect to time. There are several ways to time the control plane; we describe

3.2 Evaluation

3.2.1 Timing the Control Plane

Switch Logic

Controller Logic

3.2.2 Controller Logic

3.2.3 Controller Logic

3.2.4 Controller Logic

3.2.5 Controller Logic

3.2.6 Controller Logic

3.2.7 Controller Logic

3.2.8 Controller Logic

3.2.9 Controller Logic

3.2.10 Controller Logic

3.2.11 Controller Logic

3.2.12 Controller Logic

3.2.13 Controller Logic

3.2.14 Controller Logic

3.2.15 Controller Logic

3.2.16 Controller Logic

3.2.17 Controller Logic

3.2.18 Controller Logic

3.2.19 Controller Logic

3.2.20 Controller Logic

3.2.21 Controller Logic

3.2.22 Controller Logic

3.2.23 Controller Logic

3.2.24 Controller Logic

3.2.25 Controller Logic

3.2.26 Controller Logic

3.2.27 Controller Logic

3.2.28 Controller Logic

3.2.29 Controller Logic

3.2.30 Controller Logic

3.2.31 Controller Logic

3.2.32 Controller Logic

3.2.33 Controller Logic

3.2.34 Controller Logic

3.2.35 Controller Logic

3.2.36 Controller Logic

3.2.37 Controller Logic

3.2.38 Controller Logic

3.2.39 Controller Logic

3.2.40 Controller Logic

3.2.41 Controller Logic

3.2.42 Controller Logic

3.2.43 Controller Logic

3.2.44 Controller Logic

3.2.45 Controller Logic

3.2.46 Controller Logic

3.2.47 Controller Logic

3.2.48 Controller Logic

3.2.49 Controller Logic

3.2.50 Controller Logic

3.2.51 Controller Logic

3.2.52 Controller Logic

3.2.53 Controller Logic

3.2.54 Controller Logic

3.2.55 Controller Logic

3.2.56 Controller Logic

3.2.57 Controller Logic

3.2.58 Controller Logic

3.2.59 Controller Logic

3.2.60 Controller Logic

3.2.61 Controller Logic

3.2.62 Controller Logic

3.2.63 Controller Logic

3.2.64 Controller Logic

3.2.65 Controller Logic

3.2.66 Controller Logic

3.2.67 Controller Logic

3.2.68 Controller Logic

3.2.69 Controller Logic

3.2.70 Controller Logic

3.2.71 Controller Logic

3.2.72 Controller Logic

3.2.73 Controller Logic

3.2.74 Controller Logic

3.2.75 Controller Logic

3.2.76 Controller Logic

3.2.77 Controller Logic

3.2.78 Controller Logic

3.2.79 Controller Logic

3.2.80 Controller Logic

3.2.81 Controller Logic

3.2.82 Controller Logic

3.2.83 Controller Logic

3.2.84 Controller Logic

3.2.85 Controller Logic

3.2.86 Controller Logic

3.2.87 Controller Logic

3.2.88 Controller Logic

3.2.89 Controller Logic

3.2.90 Controller Logic

3.2.91 Controller Logic

3.2.92 Controller Logic

3.2.93 Controller Logic

3.2.94 Controller Logic

3.2.95 Controller Logic

3.2.96 Controller Logic

3.2.97 Controller Logic

3.2.98 Controller Logic

3.2.99 Controller Logic

3.2.100 Controller Logic

3.2.101 Controller Logic

3.2.102 Controller Logic

3.2.103 Controller Logic

3.2.104 Controller Logic

3.2.105 Controller Logic

3.2.106 Controller Logic

3.2.107 Controller Logic

3.2.108 Controller Logic

3.2.109 Controller Logic

3.2.110 Controller Logic

3.2.111 Controller Logic

3.2.112 Controller Logic

3.2.113 Controller Logic

3.2.114 Controller Logic

3.2.115 Controller Logic

3.2.116 Controller Logic

3.2.117 Controller Logic

3.2.118 Controller Logic

3.2.119 Controller Logic

3.2.120 Controller Logic

3.2.121 Controller Logic

Thank You!

Timing SDN Control Planes to Infer Network Configurations

- OpenFlow networks have timing side channels
- Adversaries can potentially learn fine grained information about OpenFlow networks, and their configurations, **without compromising equipment**
- There are many implications and (hopefully) defenses



University of Colorado
Boulder



Potential Defenses

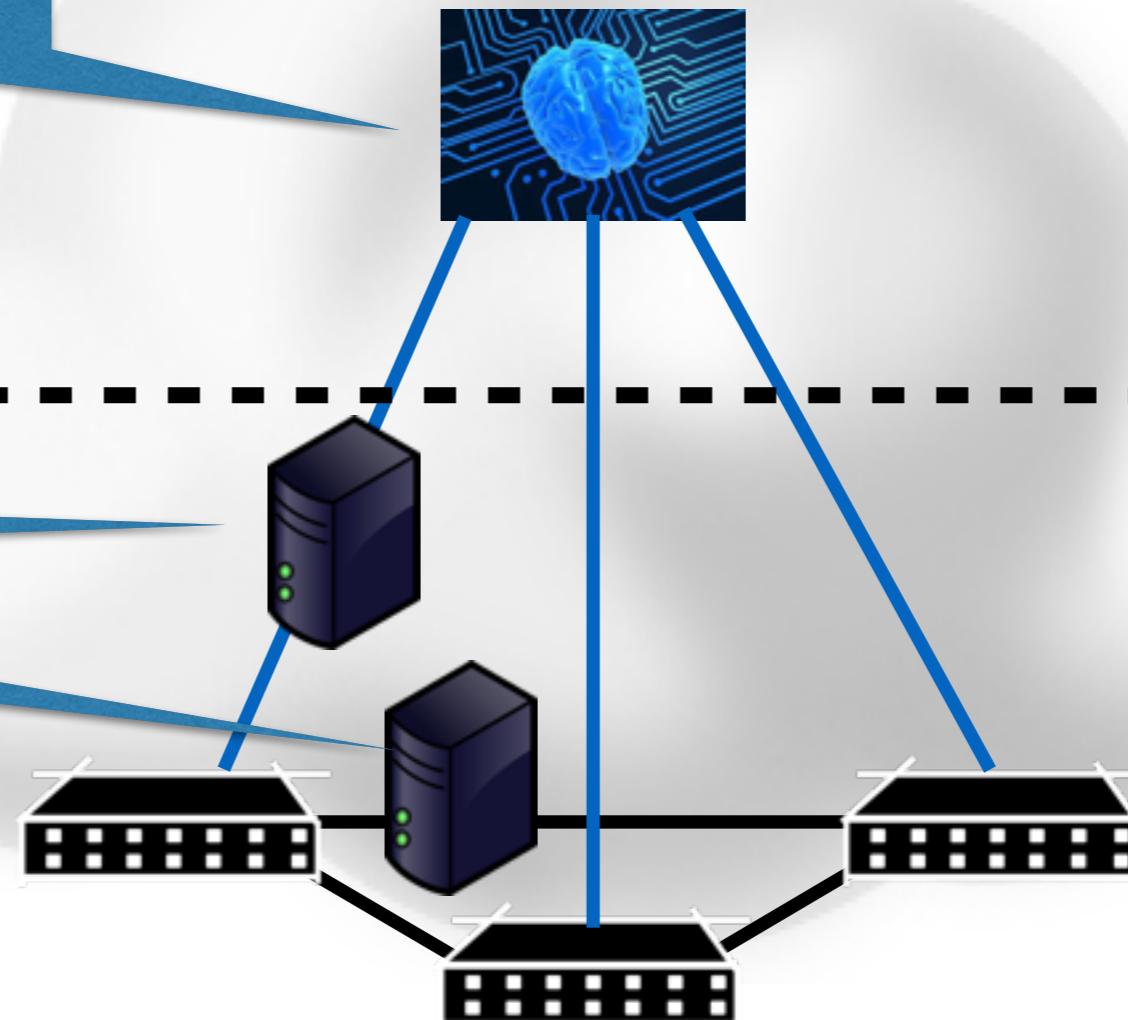
Real time controllers?

Middlebox
detectors and
filters?

Programmable
switches? (OFX, P4)



Control Plane



Data Plane