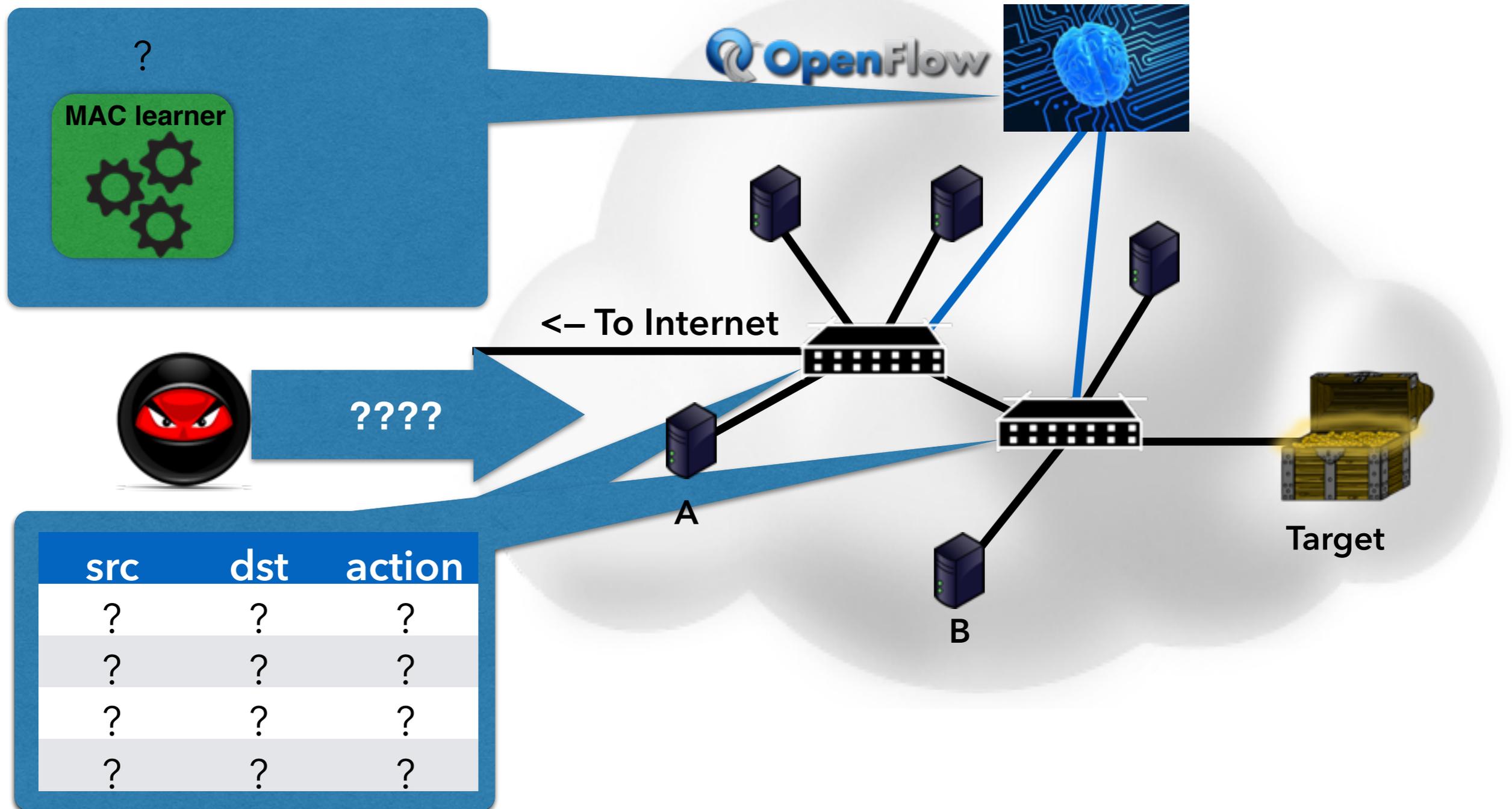


Timing SDN Control Planes to Infer Network Configurations

John Sonchack, Adam J. Aviv and Eric Keller



Attack Goal: Probe An OpenFlow Network to Learn its Configuration



Outline

Introduction

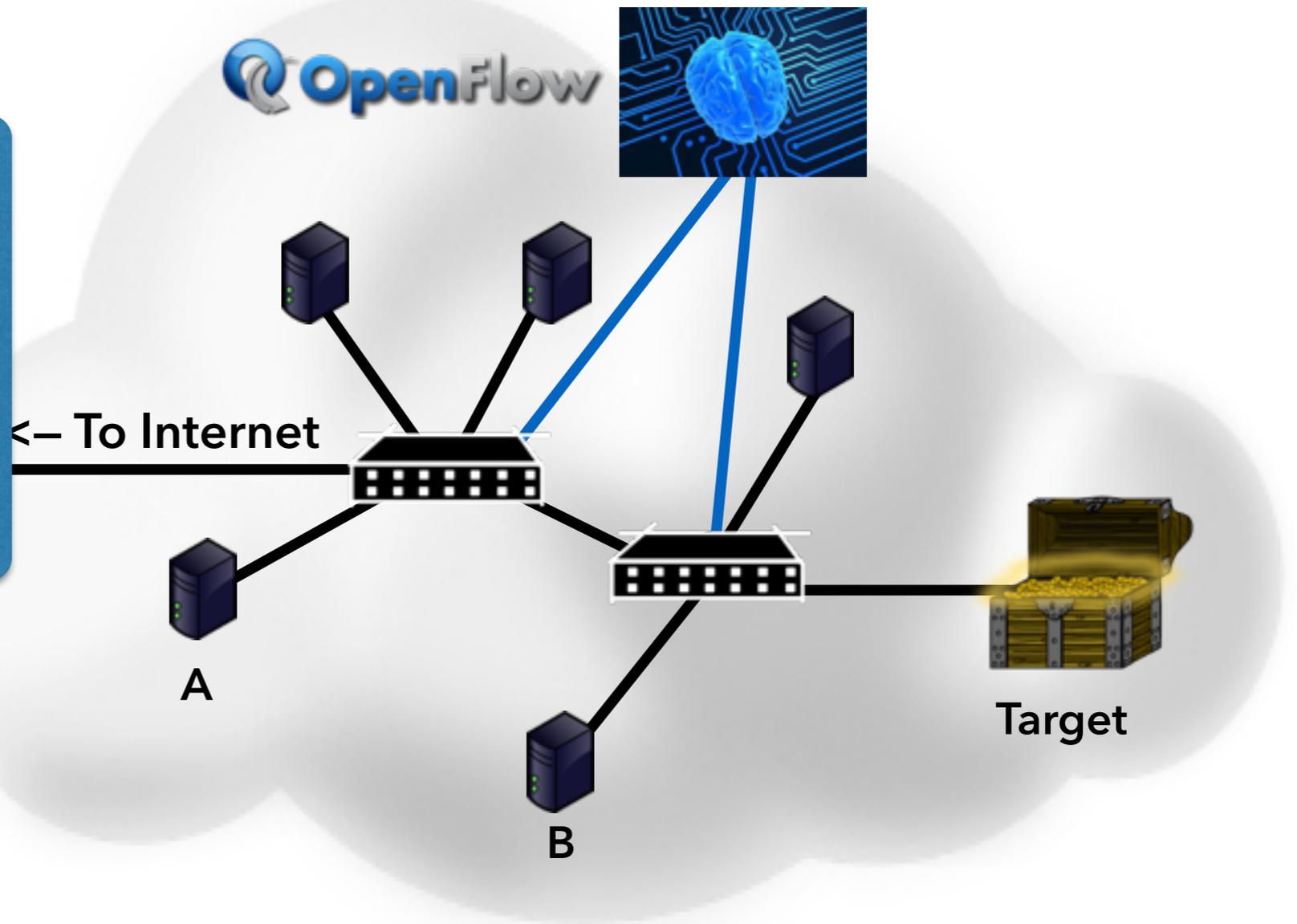
OpenFlow Timing Attacks

**A More General OpenFlow
Timing Attack**

Preliminary Results

Motivation: Plan Multi-Staged Attacks

What forwarding rules are installed on the switches?

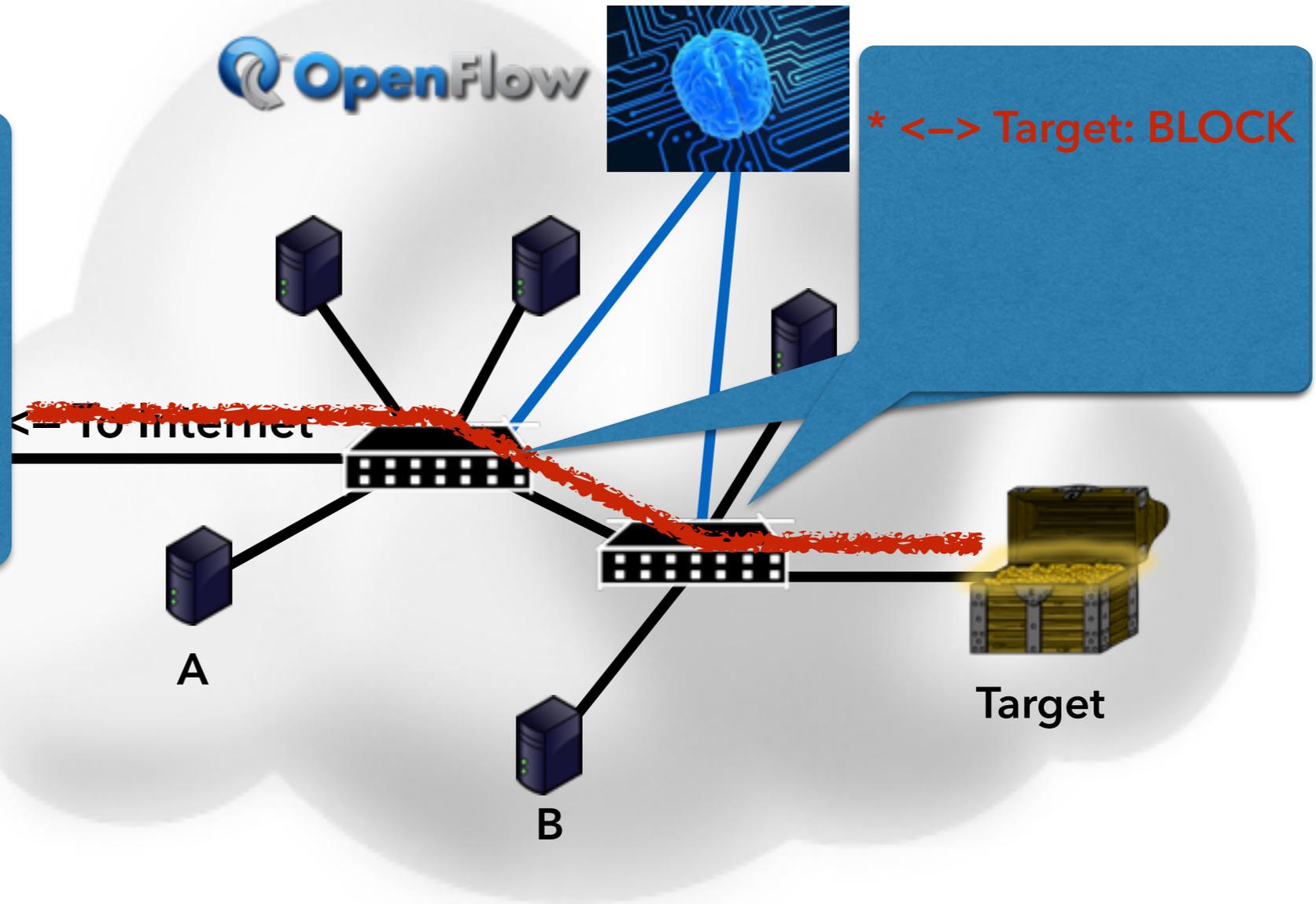


Motivation: Plan Multi-Staged Attacks

Attacks

What forwarding rules are installed on the switches?

Attack Plan:
?? → Target



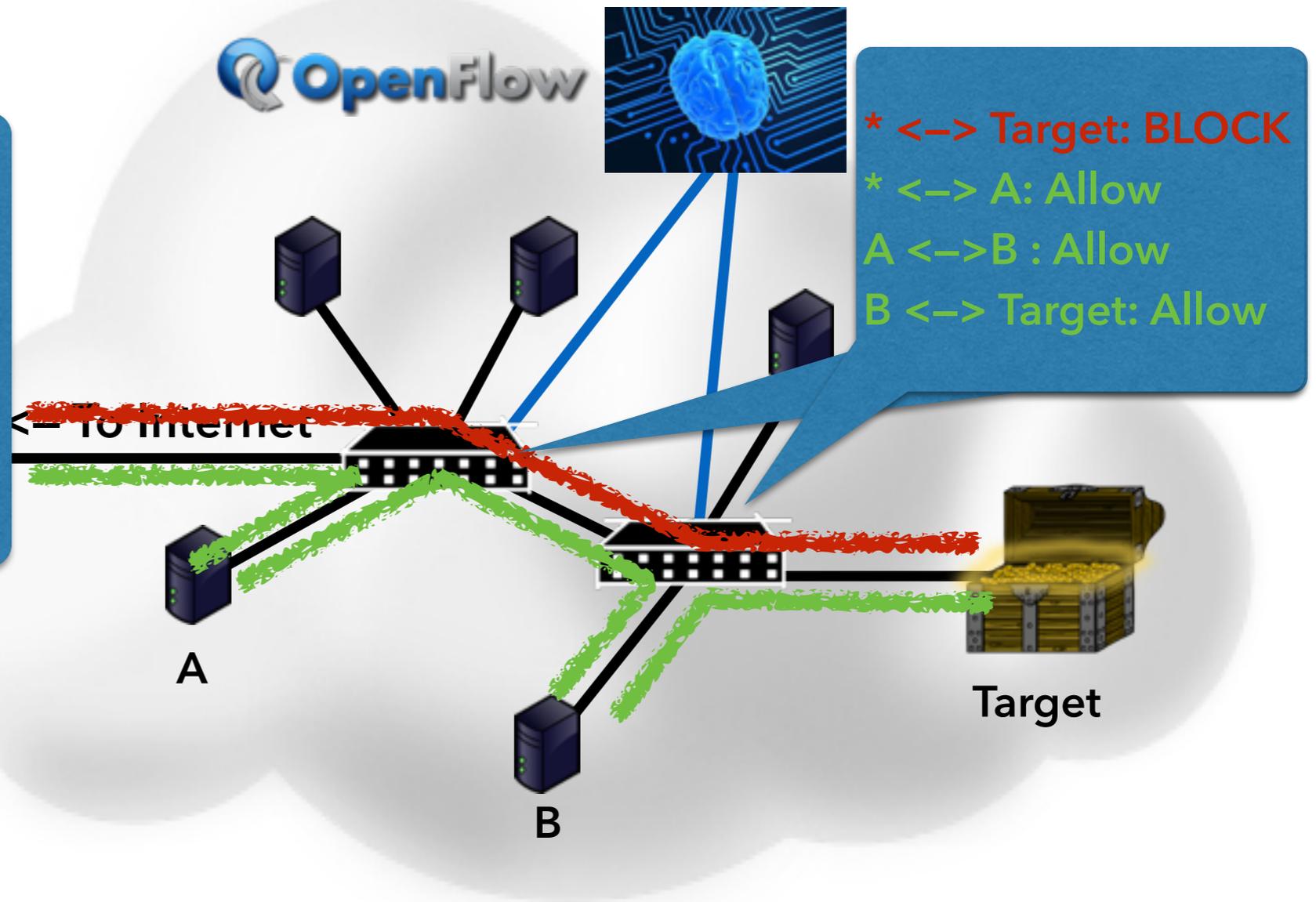
Motivation: Plan Multi-Staged Attacks

Attacks

What forwarding rules are installed on the switches?

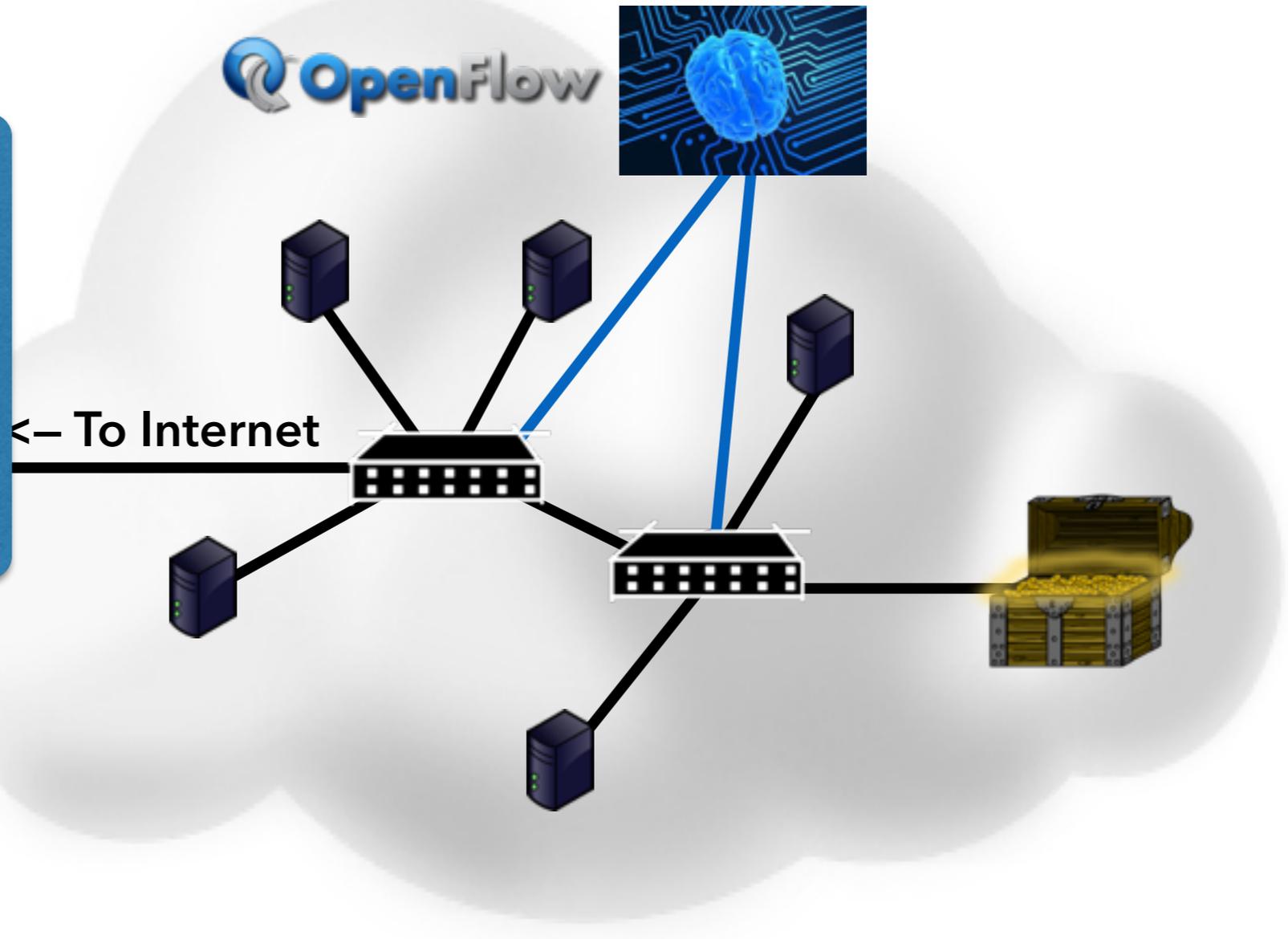
Attack Plan:

A → B → Target



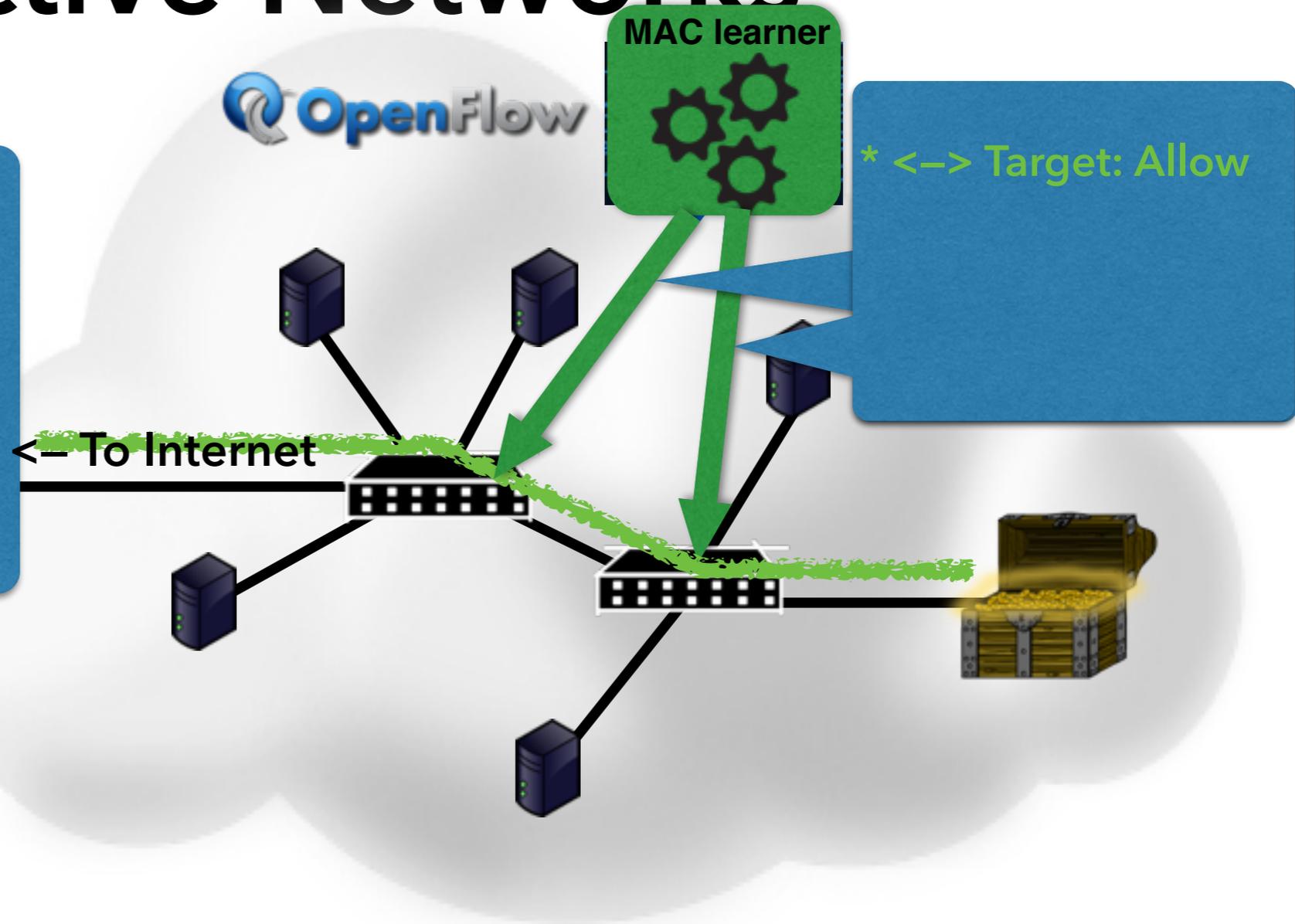
Motivation: Reprogramming Reactive Networks

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

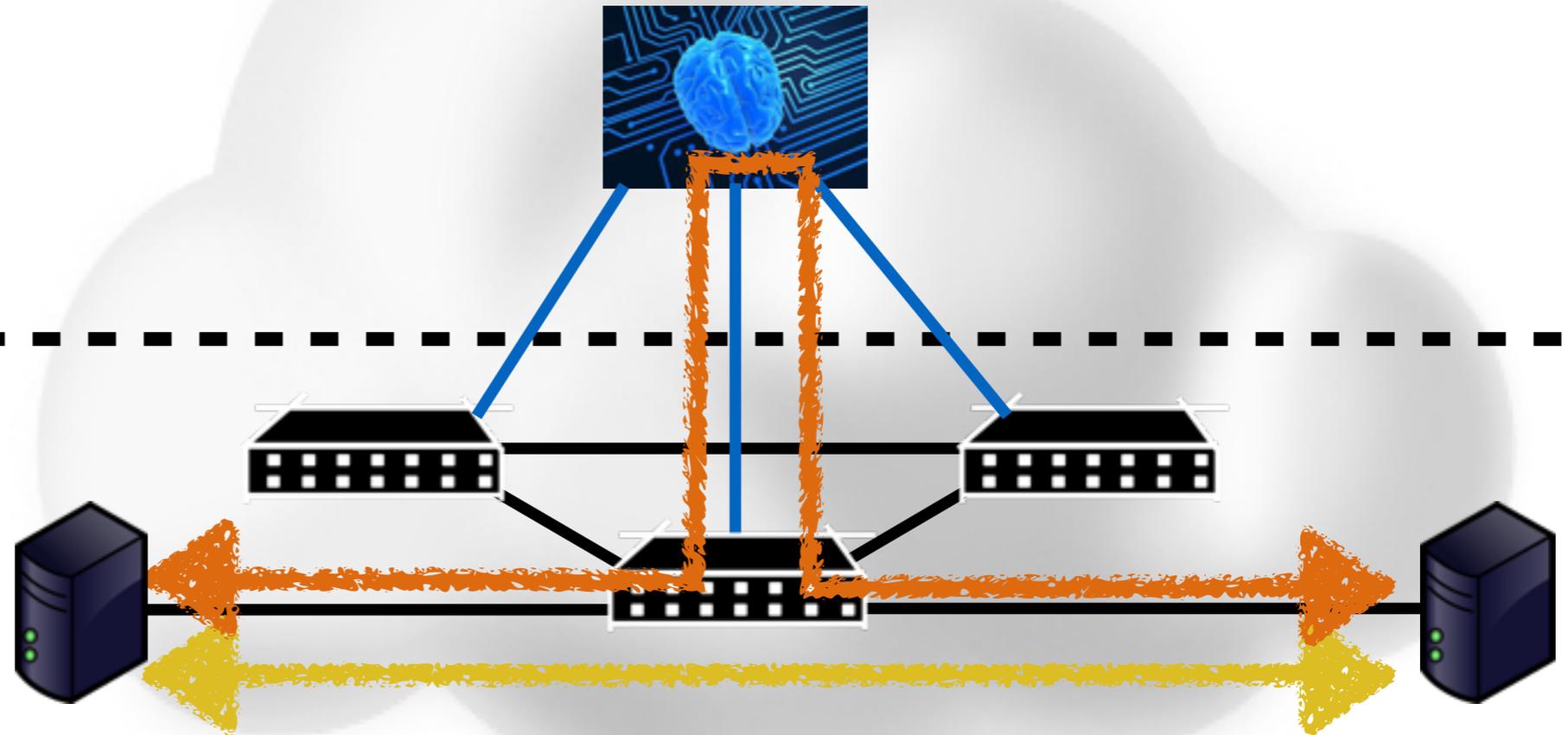
**A More General OpenFlow
Timing Attack**

Preliminary Results

A Simple OpenFlow Timing Property



Control Plane



Data Plane

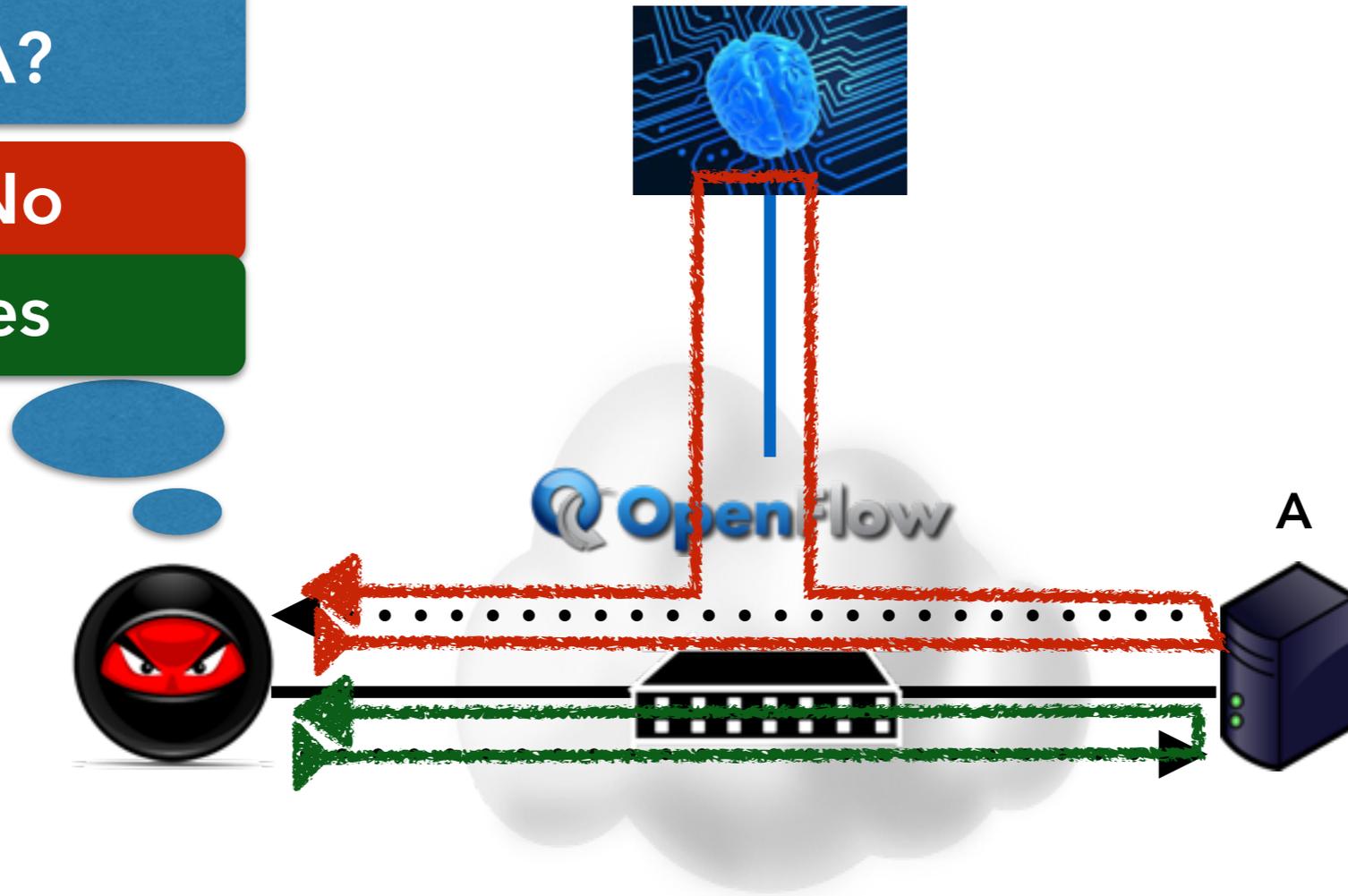
```
PING 1.1.1.2 (1.1.1.2) 56(84) bytes of data.  
64 bytes from 1.1.1.2: icmp_seq=1 ttl=64 time=2.56 ms  
64 bytes from 1.1.1.2: icmp_seq=2 ttl=64 time=0.345 ms  
64 bytes from 1.1.1.2: icmp_seq=3 ttl=64 time=0.044 ms
```

A Simple OpenFlow Timing Attack

Rule installed from
My host \leftrightarrow A?

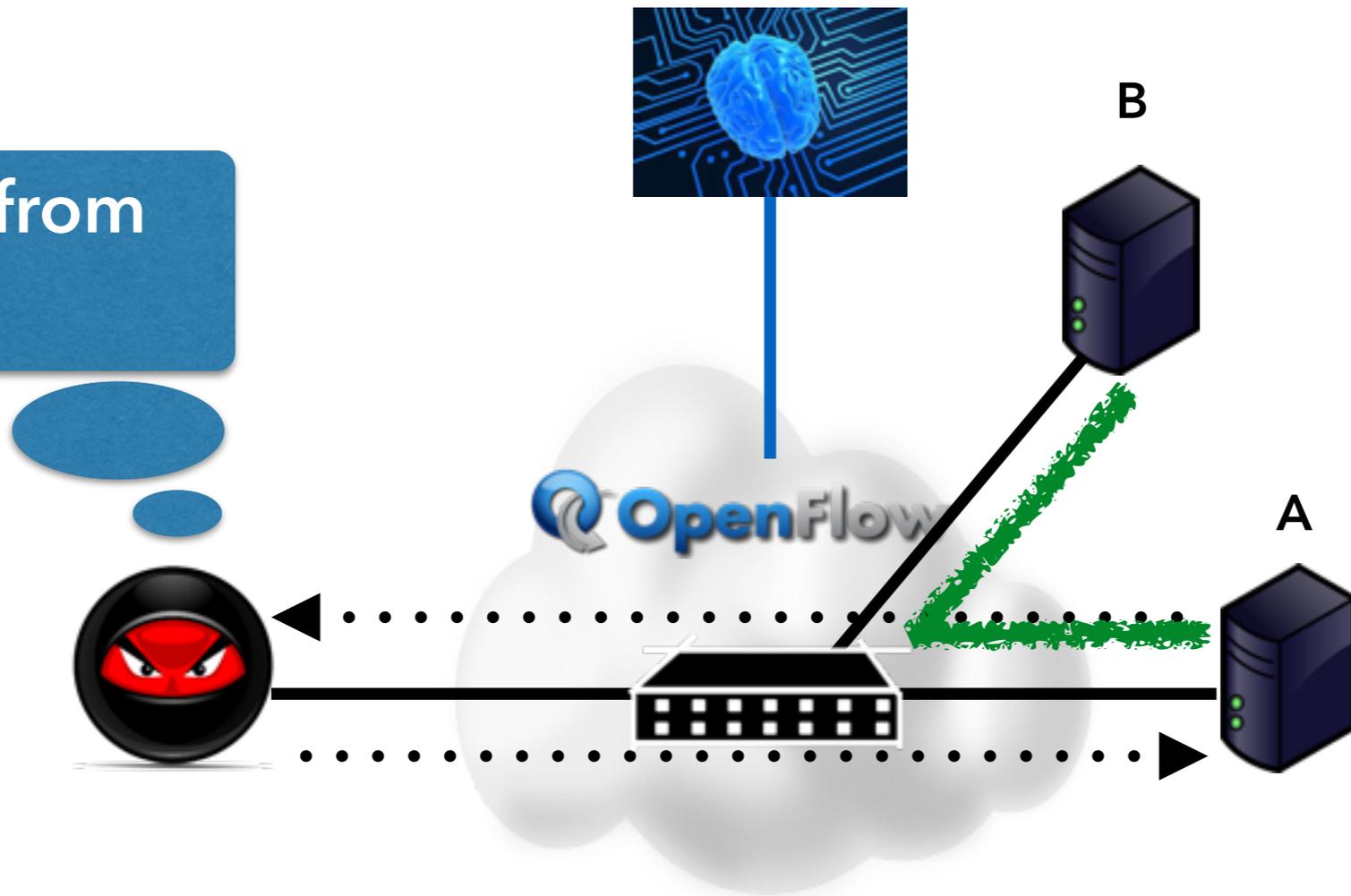
High RTT \rightarrow No

Low RTT \rightarrow Yes

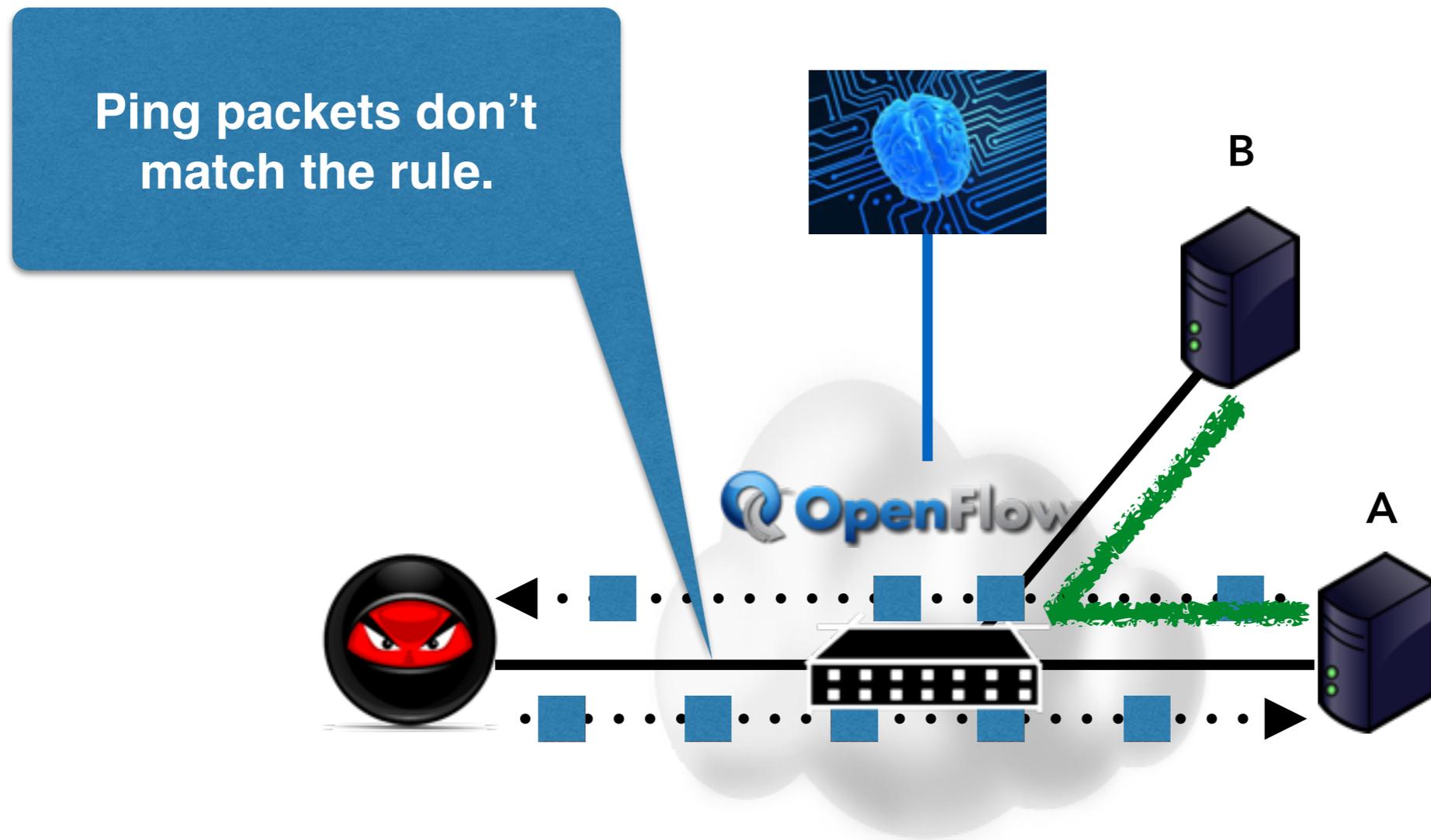


A Simple OpenFlow Timing Attack: Limitations

Rule installed from
A -> B?

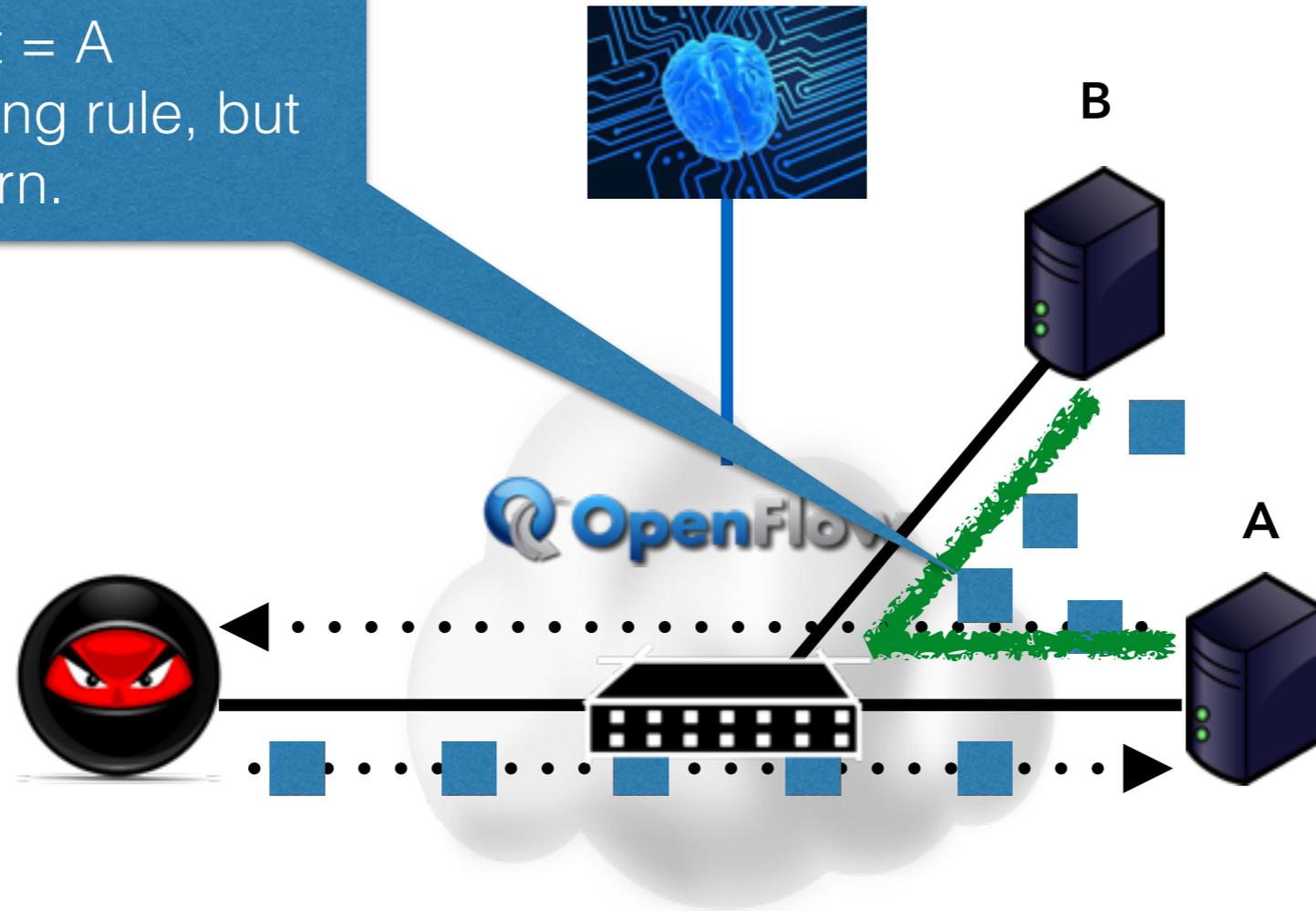


A Simple OpenFlow Timing Attack: Limitations



A Simple OpenFlow Timing Attack: Limitations

replies to spoofed packets with
 $\text{src} = B$, $\text{dst} = A$
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]

NETWORK PROTOCOLS (ICNP), 2013 2131
IEEE International Conference on,
pages 1–6. IEEE, 2013.

preprint arXiv:1504.03095,
2015.

Outline

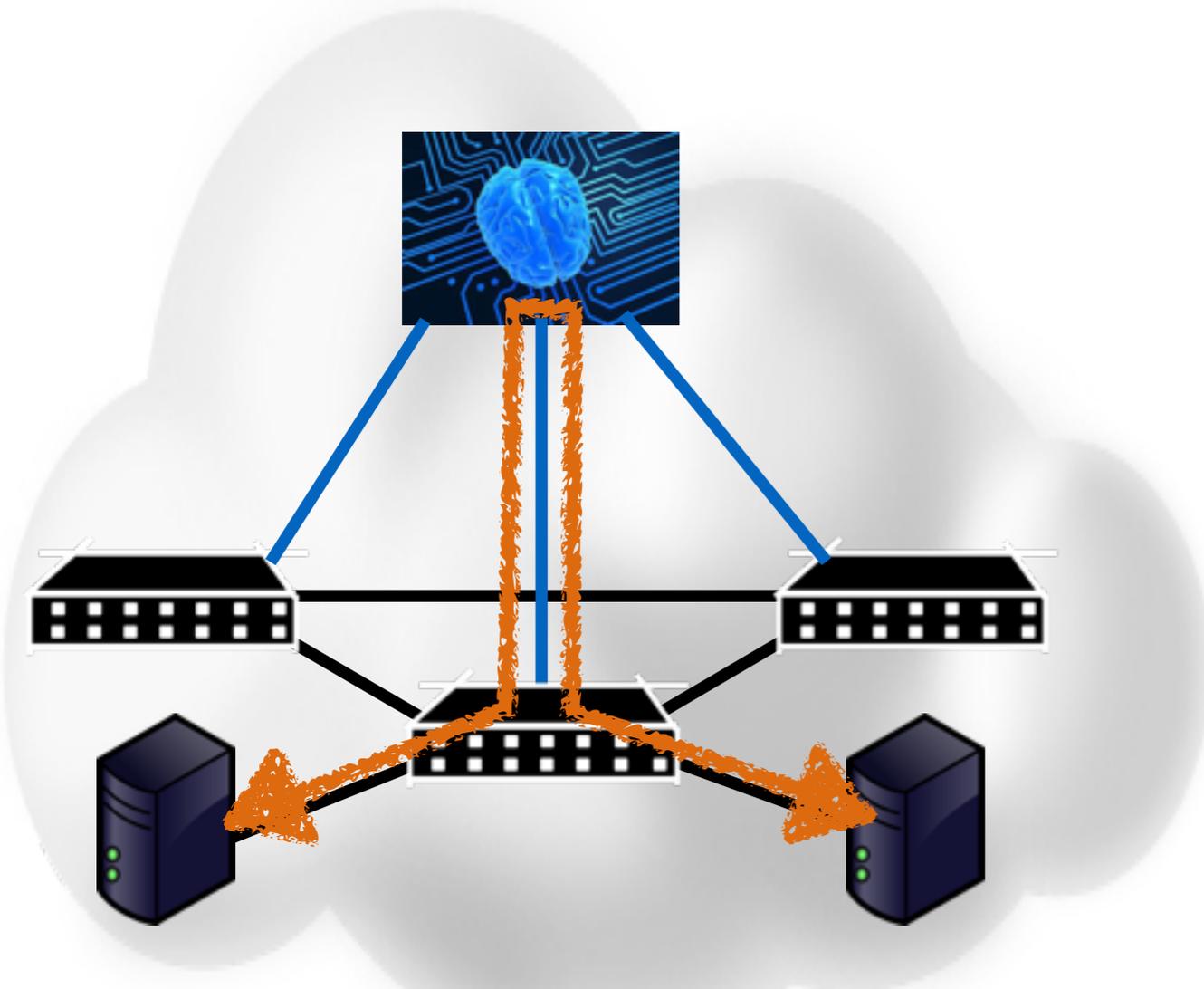
Introduction

OpenFlow Timing Attacks

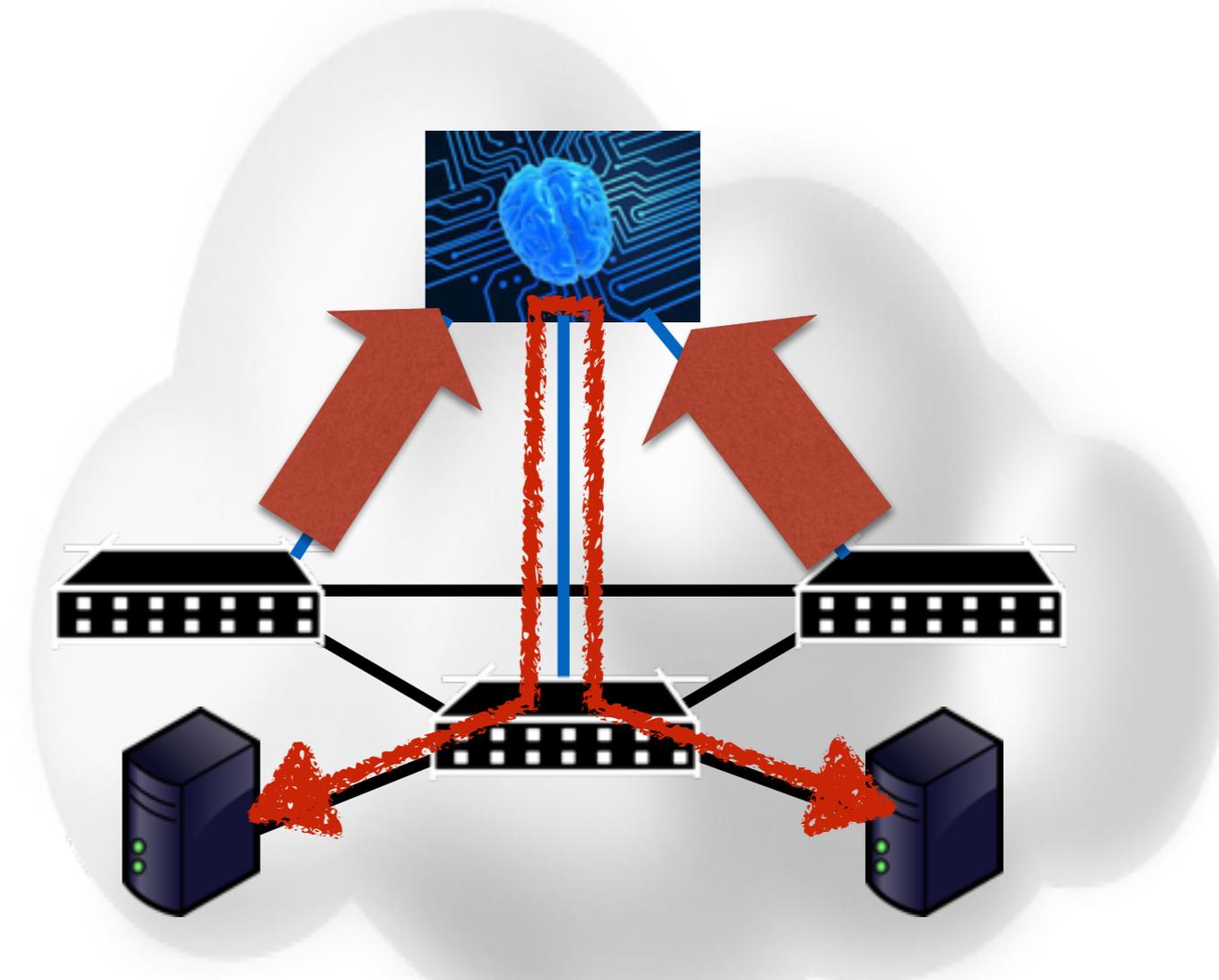
**A More General OpenFlow
Timing Attack**

Preliminary Results

OpenFlow Timing Property: Control Planes Response Time Depends on Load



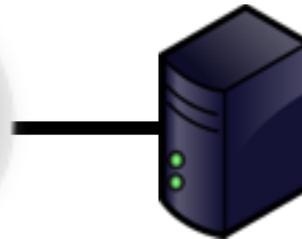
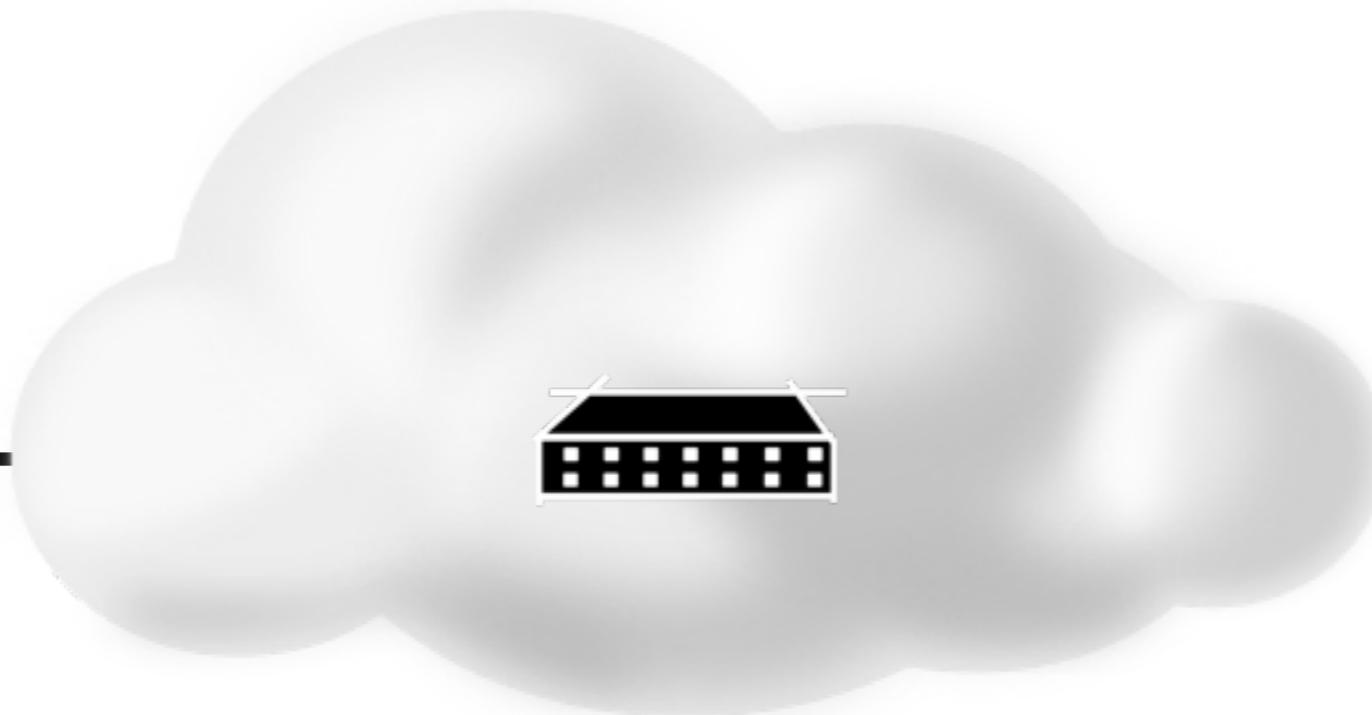
```
PING 1.1.1.2
64 bytes from 1.1.1.2: time=2.56 ms
64 bytes from 1.1.1.2: time=0.345 ms
64 bytes from 1.1.1.2: time=0.044 ms
```



```
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
```

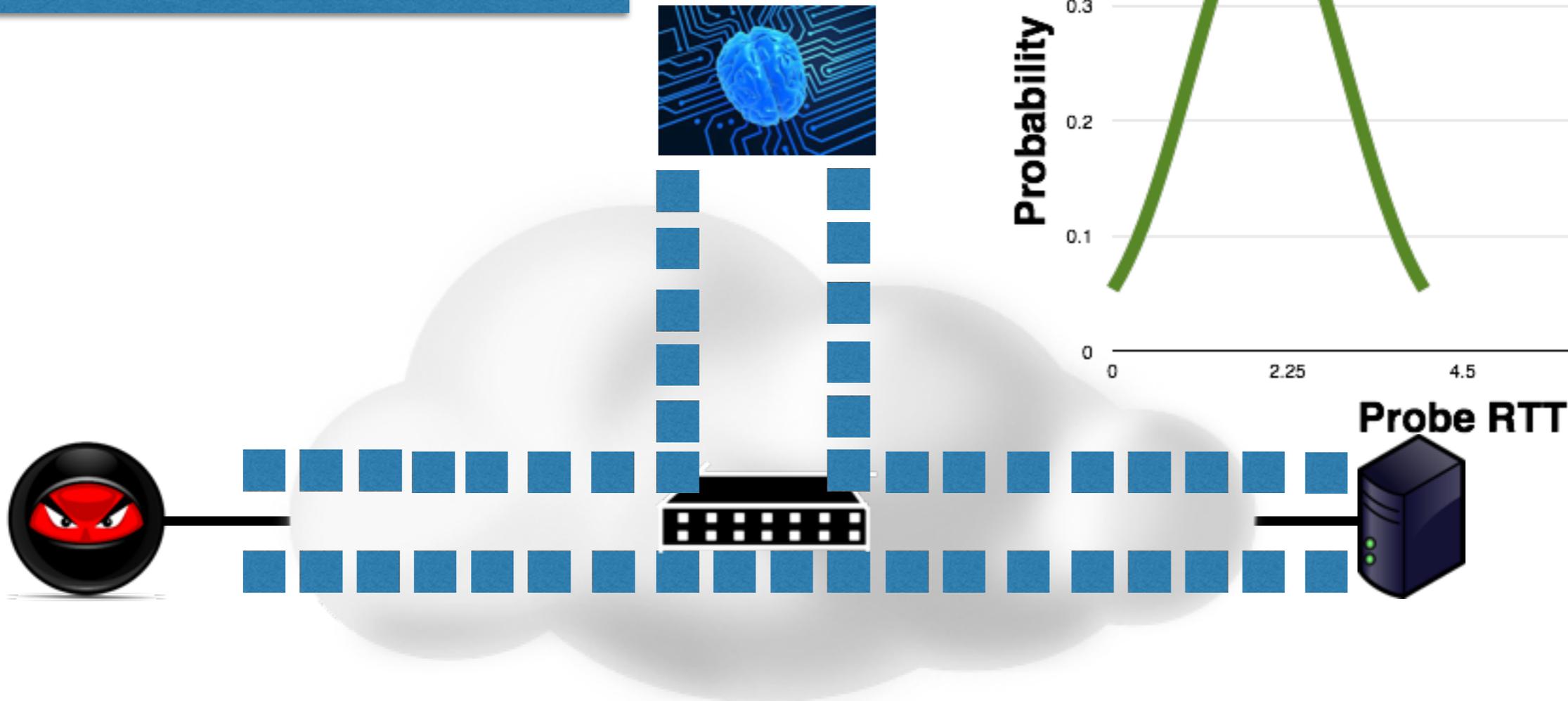
The Basic Approach: Estimating Control Plane Load Change

Will these packets match a rule?



The Basic Approach: Estimating Control Plane Load Change

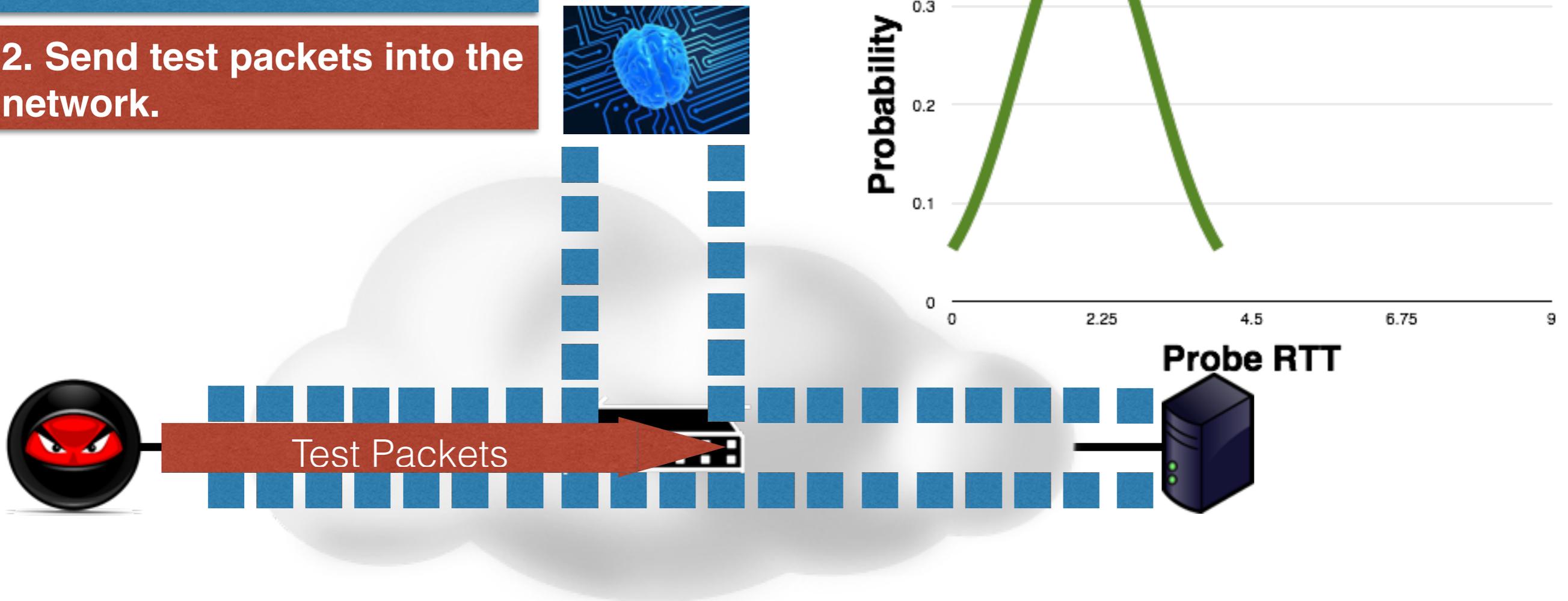
1. Send timing probes through the control plane, measure RTT.



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

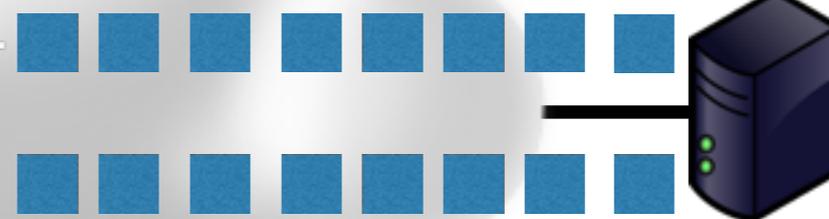
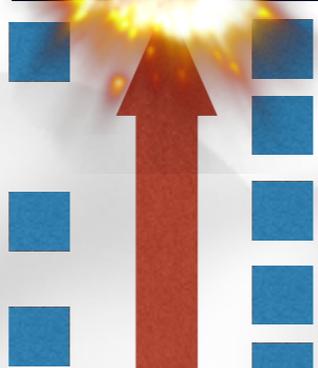
1. Send timing probes through the control plane, measure RTT.

2. Send test packets into the network.

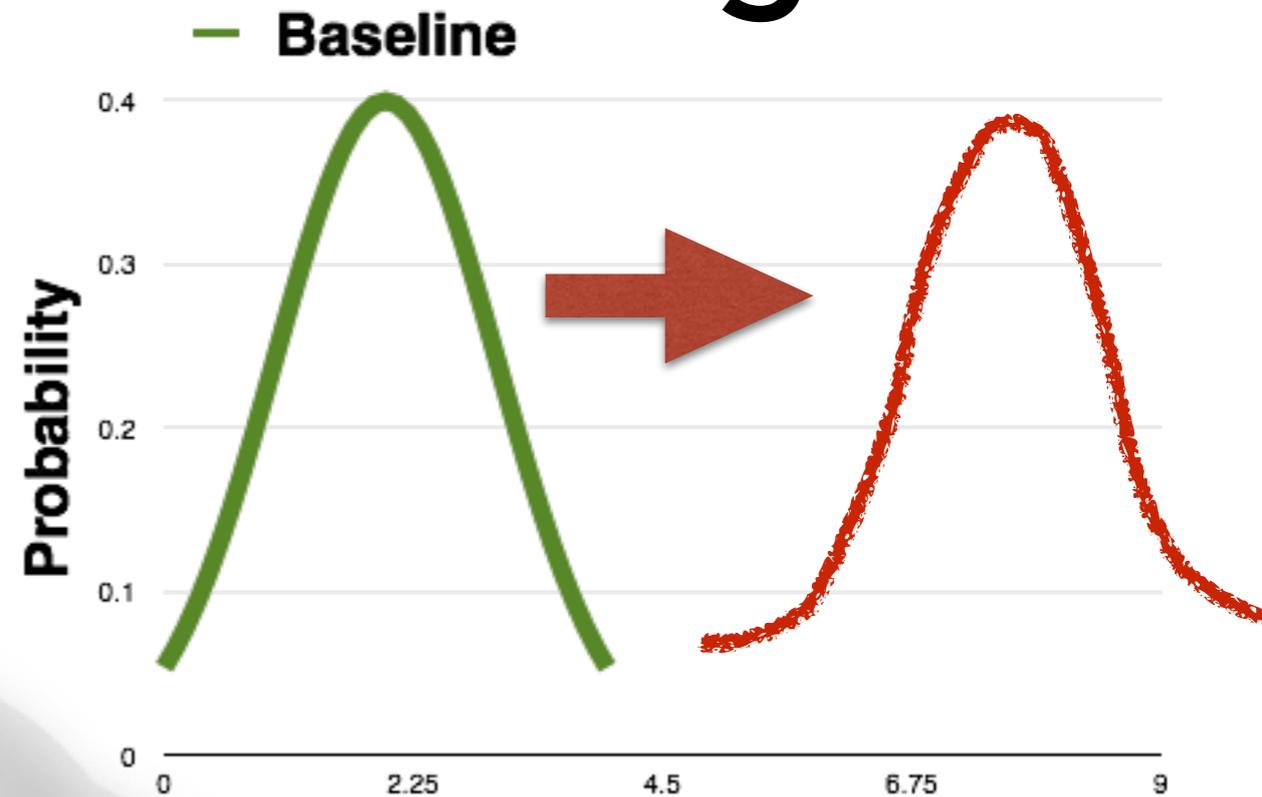
3. If the packets reach the controller, its load will increase, delaying the probes.



Test Packets



Probe RTT



The Basic Approach: Estimating Control Plane Load Change

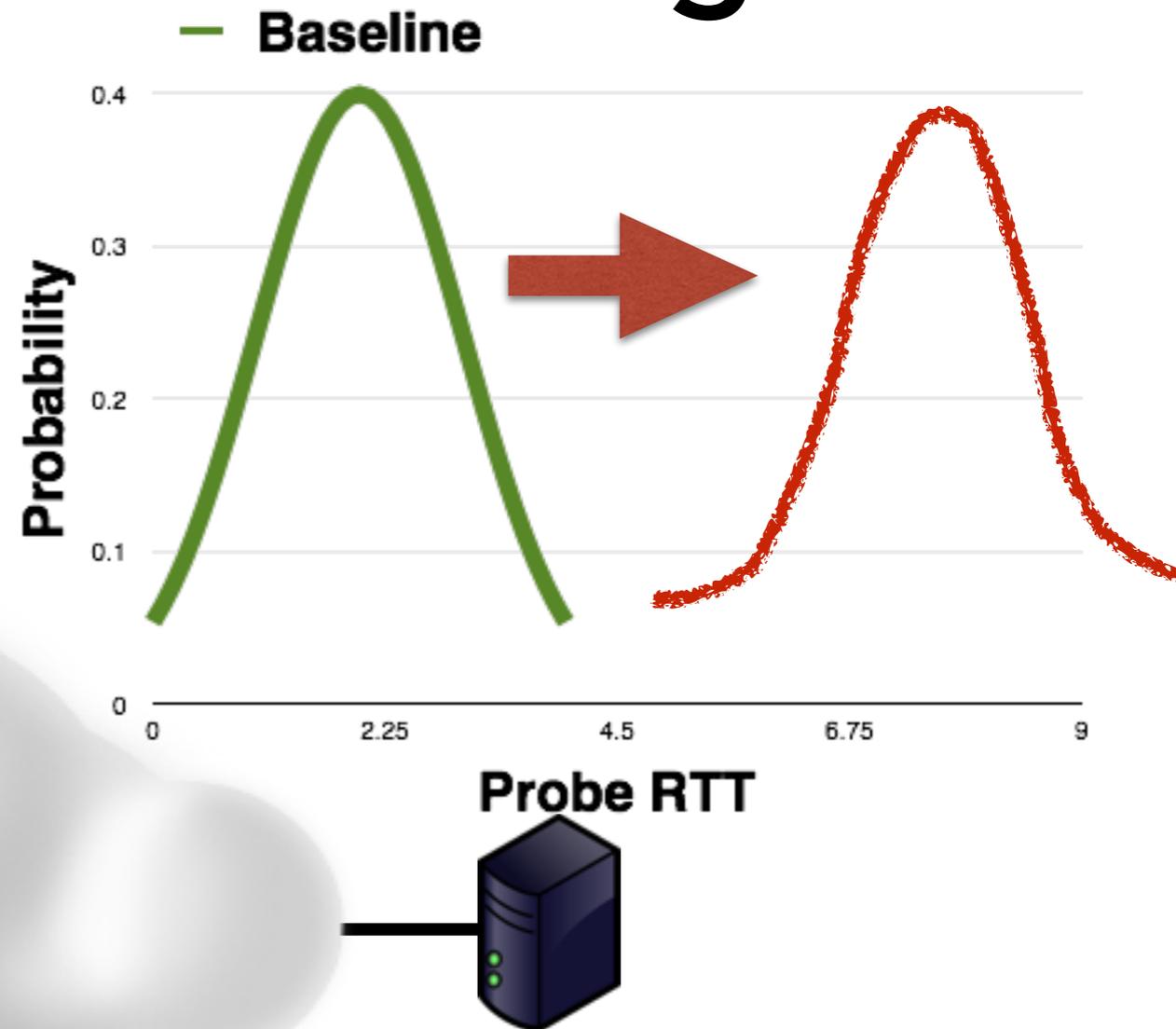
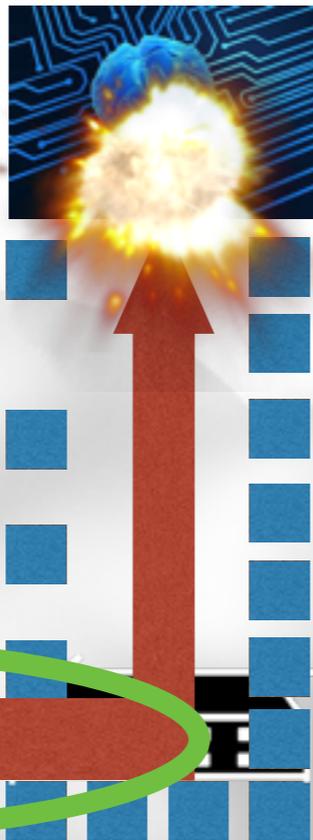
1. Send timing probes through the control plane, measure RTT.

2. Send test packets into the network.

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



Test Packets



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

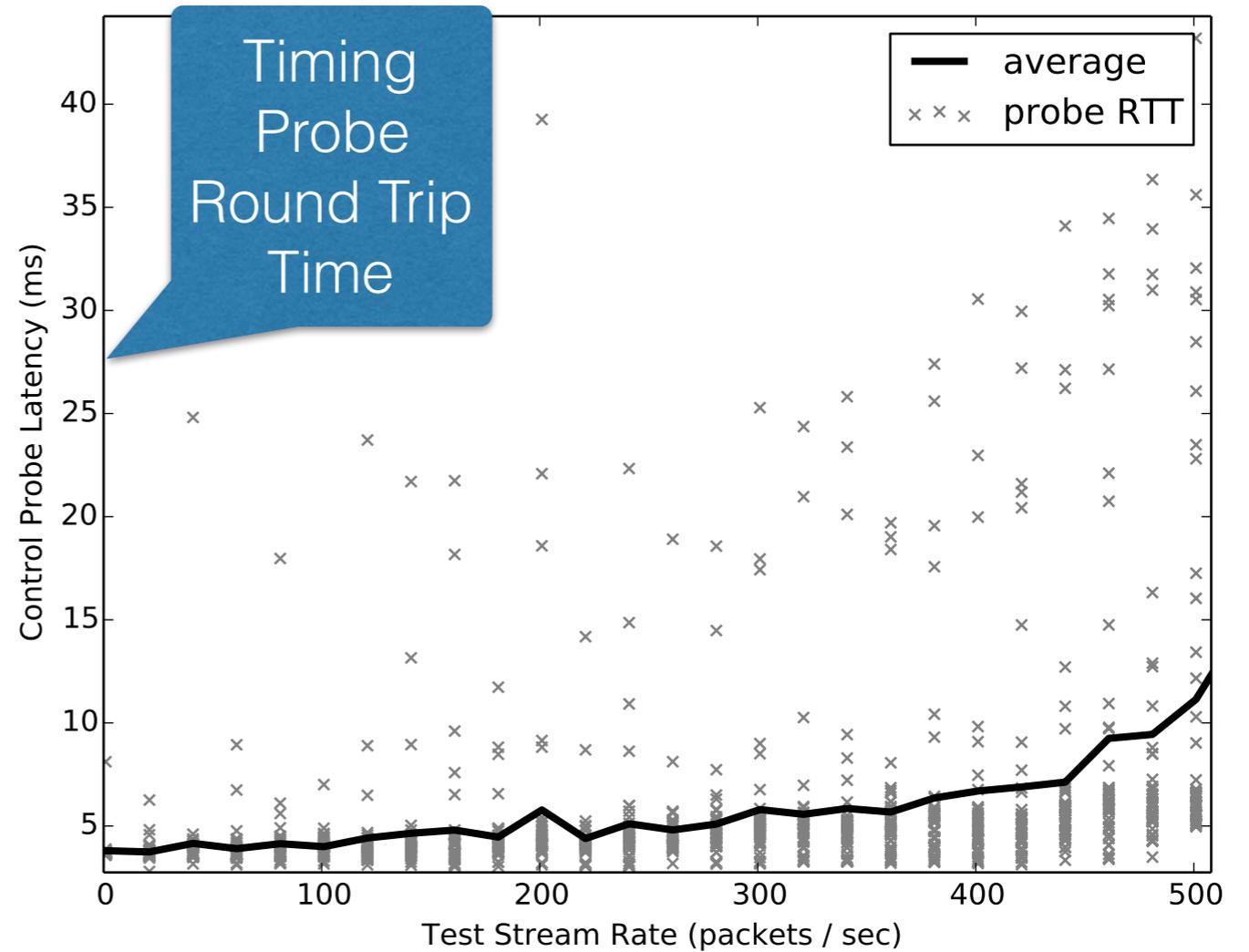
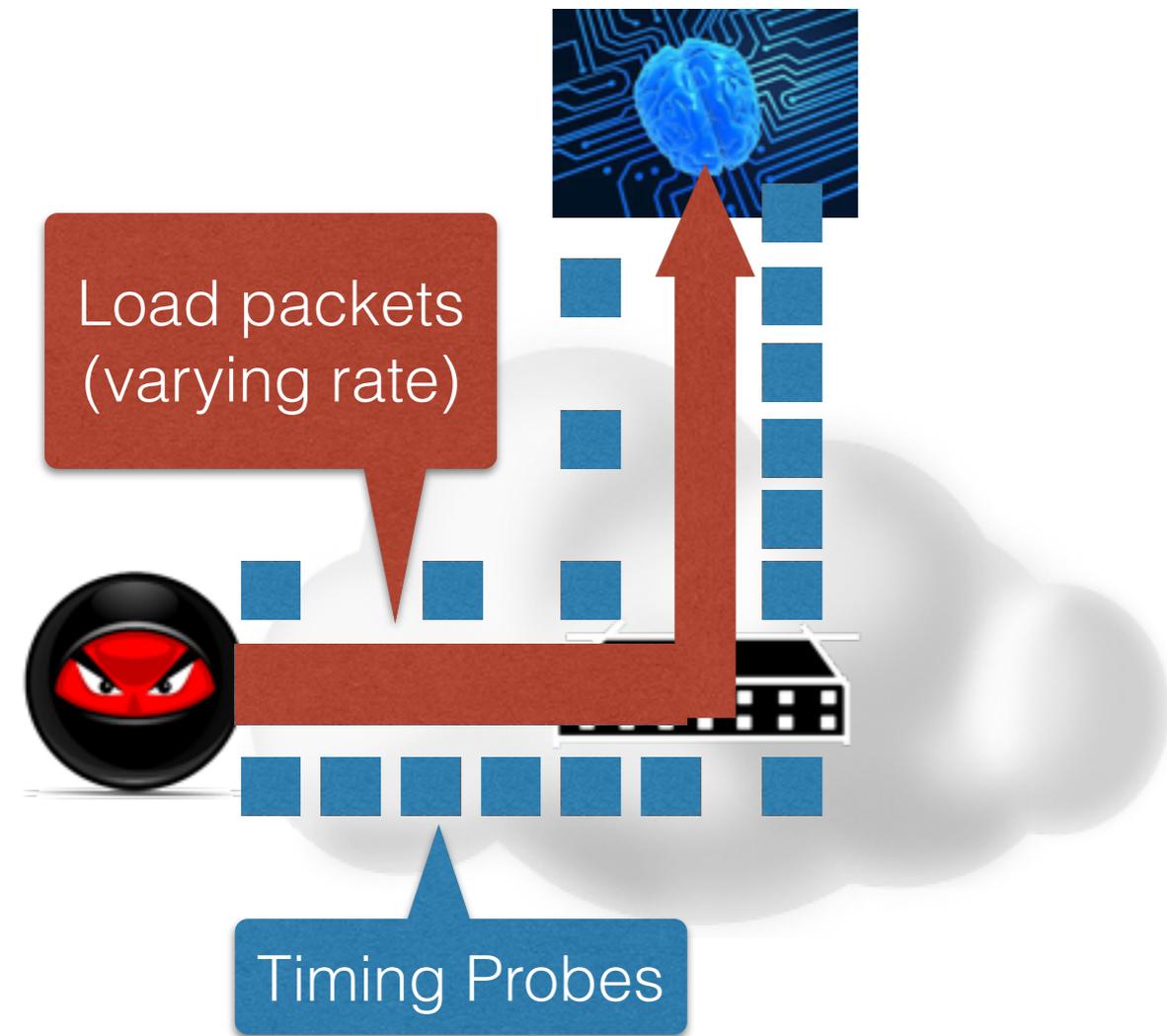
Preliminary Results

Preliminary Results

Does probe RTT estimate controller load?

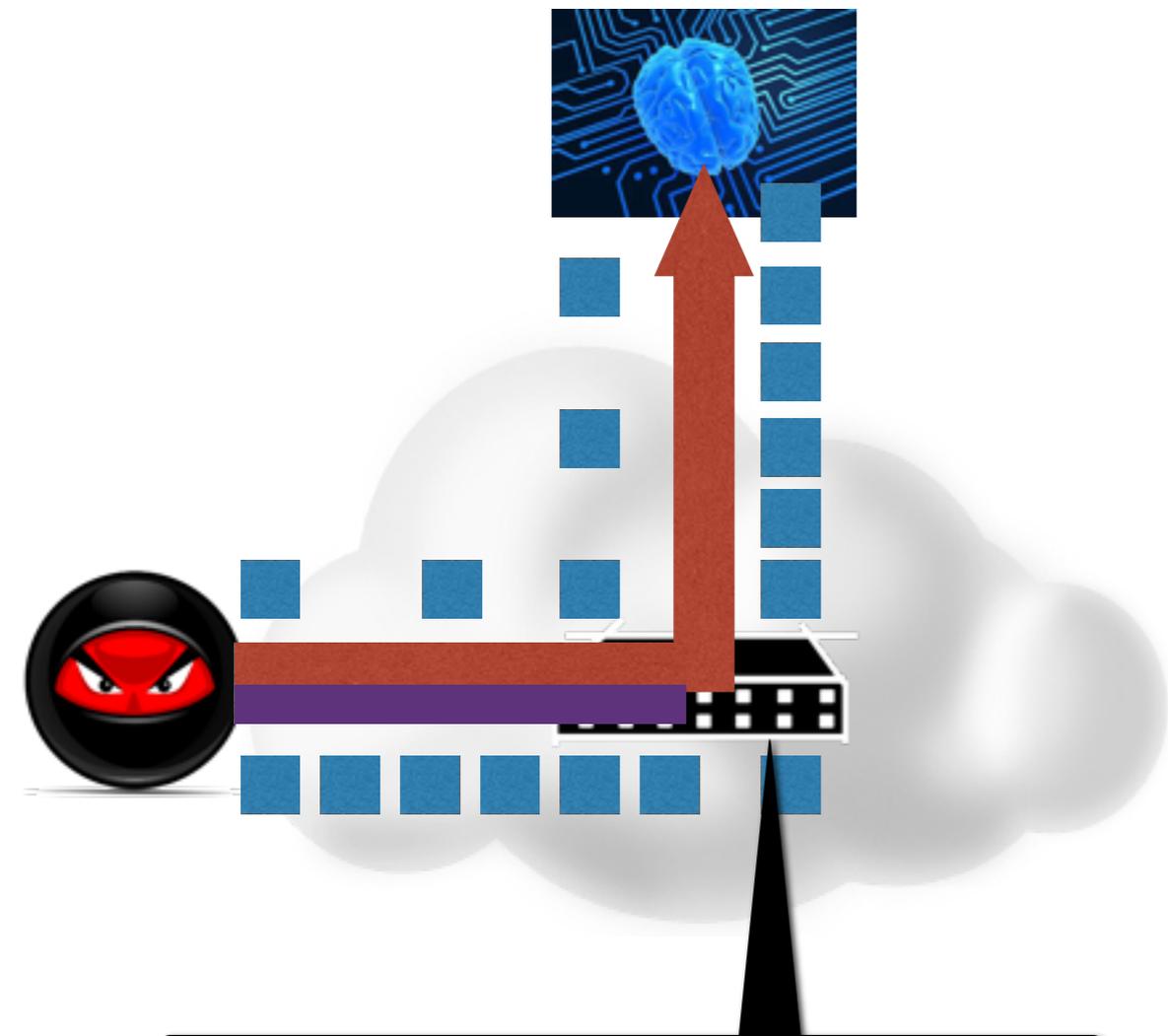
Can an adversary learn if a forwarding rule is installed?

Estimating Controller Load



Packets per second sent to controller

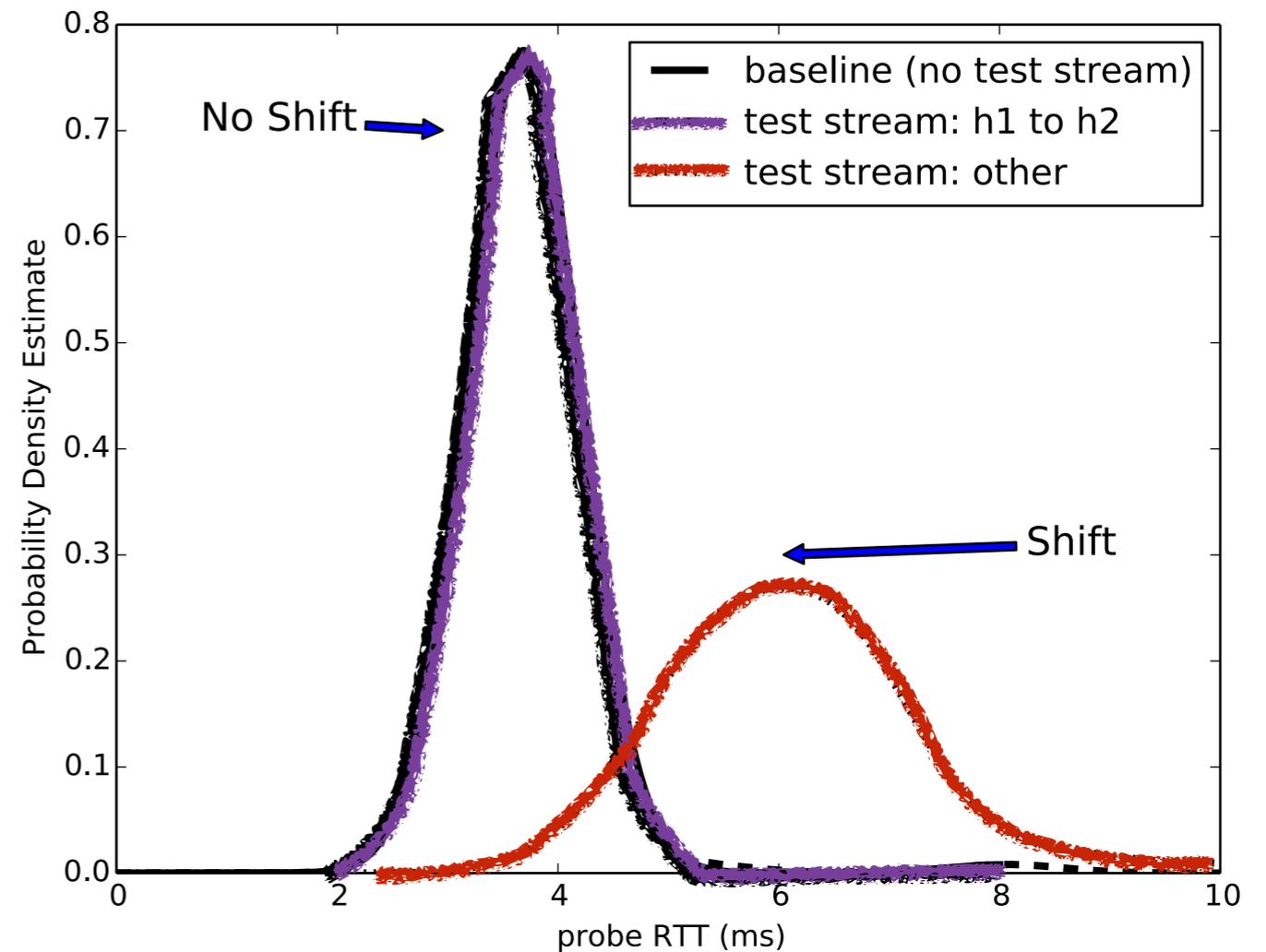
Learning if a Rule is Installed



Installed Rules:

src:h1 dst:h2 → *output port 2*

DEFAULT → send to controller



Timing Probe Round Trip Time

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 Networks. We find that the response time of an OpenFlow control plane depends on its workload, and we develop 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 transform the way networking is done by opening up the interfaces to network elements in a programmable way, and organizations such as Facebook, Google, and the NSA have already deployed large scale SDN networks [1]. A key aspect of SDN is the separation of the control and data planes in a network. The data plane forwards data across the network as far as possible by matching packets against simple forwarding rules, while the control plane installs rules into the data plane and performs more advanced packet processing as needed (e.g., when a packet cannot determine how to 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 leak information about a network. Previous work showed that by measuring the response time of servers in a SDN, adversaries can determine whether or not they require rules through the control plane and infer the network's forwarding table contents and whether it contains links that aggregate flows [2].

In this paper, we study a more sophisticated inference attack that an adversary can use to infer much more about a network. Our new observation is that how long the control plane takes to process a packet depends on how much load is on the control plane. An adversary can time the control plane while injecting packets into an SDN to determine whether the injected packets are reaching the control plane or causing the control plane to install new forwarding rules even if the packets do not reach a switch from being in the SDN. In contrast, previous SDN inference attacks could only determine whether legitimate requests to servers in the SDN went through the control plane. By measuring this statistic, this new attack allows adversaries who control only a single host in an SDN to 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 sequence of packets causes the controller to install forwarding rules. Once adversaries have this, they can exploit switch forwarding tables and greatly degrade the victim network's performance by only sending a relatively small number

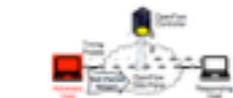


Figure 1: A summary of the control plane inference attack: an adversary sends probes that travel through the control plane while sending test packets into the network. If the control plane processes the test packets the probe RTT will increase.

of packets into the network.

In the remainder of this paper, we explore the inference attack in greater detail, and then demonstrate its feasibility on a real world with real OpenFlow switches and controllers.

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 switches and controllers. This model reflects two scenarios: first, an adversary performing a multi-stage attack that has just gained access to one host in a network through malware or social engineering and now wishes to plan subsequent stages; second, a user of a shared network or data center who wants to attack other users.

In carry out the inference attack, which Figure 1 summarizes, the adversary sends two streams of packets into the network simultaneously: a stream of timing probes and a stream of timing packets. The timing probes travel through the control plane, and their round trip time (RTT) depends on the controller's load. The timing packets are opened packets that all have the same value for one or more packet header fields. If the probe RTT increases, the adversary can infer that the control plane processed the timing packets and learn about the network. For example, if probe RTT increases when the adversary sends a stream of test packets that all have random source MAC addresses and a fixed destination MAC address, the adversary can infer that the control plane is processing all packets destined for that address, regardless of the source address.

2.1 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 load. There are several ways to time the control plane; we describe

Test Packet Streams
`test_packet_streams := {template, size, max_rate, port}`
Stream Templates
`template := {header_field = field_value, ...}`
Header Fields
`header_field := mac_source | mac_dest | ip_source | ip_dest | ...`
Header Values
`field_value := C | Ri` each packet in the stream will have the same constant value C for this field
`1 + Ri` each packet in the stream has a random value for this header field
`iC` the header field value for the *i*th packet in the stream will be *i*C

Figure 2: Syntax for test packet streams.

two below.

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 RTT of echo responses can be used to time the control plane.

Spooled ARP Requests In an OpenFlow network, MAC learning is implemented in the control plane. An OpenFlow switch sends the first packet from each 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 queued through the control plane, and the RTT of ARP requests/replies can be used to time the control plane. We use this technique in the evaluation in Section 3.

2.2 Testing the Data Plane

Test streams are streams of spoofed packets that the adversary injects 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 sending one or more packet header fields to the same value for all packets in a stream, the adversary can learn which classes of traffic get processed by the control plane.

For example, if an adversary wanted to learn whether there was 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 stream, then the controller must be processing the test stream, implying that no rule is installed. If the RTT of the timing probes does not increase, the controller must not be processing the test stream, implying that a matching forwarding rule is installed.

We specify test packet streams using the syntax in Figure 2. A test packet stream has a template that determines the header values of each packet in the stream, a size that specifies how many packets are in the stream, and a constant rate that specifies how quickly the stream should be sent into the network in packets per second.

3. EVALUATION

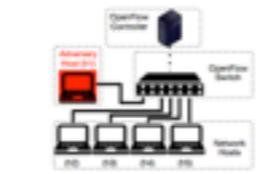


Figure 3: A diagram of our testbed network.

In this section, we evaluate the inference attack on a physical OpenFlow testbed. We focus on the following questions:

- Can an adversary determine if test stream packets are reaching the control plane?
- Can an adversary determine if the controller is installing forwarding rules in response to test packets?
- What higher level properties can an adversary learn about a network using this inference attack?

The Testbed Figure 3 illustrates the testbed network for our experiments. It contains a network infrastructure consisting of a hardware OpenFlow switch, a Pkts 1000 with an Broadcom FastPath-3 forwarding engine that processes packets in hardware according to OpenFlow rules, a K21 Mita PowerPC CPU, and 112MB of memory, and runs Debian 7, a control server, a quad core Intel i7 machine with 8GB of RAM, running Ubuntu 14.04 server LTS and the Ryu OpenFlow controller [2]. We connected 3 hosts to the switch, which we refer to as h_1 , through h_3 . We configured the hosts to have MAC addresses of 00:00:00:00:00:01 through 00:00:00:00:00:03, IP addresses of 1.1.1.1 through 1.1.1.5, and were connected to the physical ports 1 through 3 on the switch, respectively. Each host was a dual-core Intel Core 2 Duo machines with 2GB RAM. All network connections (i.e. switch to controller and switch to host) were via gigabit ethernet.

The Adversary The adversary controlled host h_1 , and could send arbitrary new packets into the network. To time the control plane, the adversary sends ARP requests to host h_2 , at a rate of 4 per second, using the technique described in Section 2.

Controller Logic The OpenFlow controller runs a simple MAC learning algorithm. When the network starts up, the controller installs a low priority rule into the switch that sends each packet up to the controller as an OpenFlow packet_in message that includes the ID of the port where the packet entered the switch. The control application keeps a map from MAC addresses to ports, which it fills using the source MAC address and input ports of the packet_in messages from the switch. When the controller receives a packet_in, it also checks the MAC table for the destination MAC of the packet. If the table contains the address, the controller installs a rule into the switch that forwards all packets with that MAC address out of the associated port. Otherwise, the controller instructs the switch to flood the packet.

Switch Logic To bootstrap the switch, we preinstalled the forwarding rules depicted in Table 1. This results a scenario where some forwarding rules are installed (either by the controller or by the network operator) before the adversary gains access to the host in the network.

Source MAC	Destination MAC	Action	Priority
00:00:00:00:00:01	00:00:00:00:00:01	Output on port 1	High
00:00:00:00:00:02	00:00:00:00:00:02	Output on port 4	High
00:00:00:00:00:03	00:00:00:00:00:03	Send to controller	Low

Table 1: The testbed switch's initial forwarding table. All packet header fields not shown are set to wildcards.

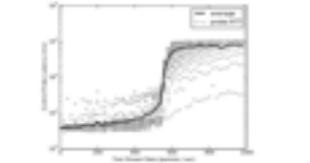


Figure 4: Timing probe RTT at test stream packet rate rates, for test streams where each packet invokes a rule installation.

3.1 Are packets reaching the controller?

To determine the effect of packets reaching the control plane on probe RTTs, we measured probe RTTs while sending a test packet stream into the network from h_1 , that only matched the default send to controller rule on the switch. We used the packet stream template: `mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:01`. Figure 4 shows probe RTT as the rate of the test stream varied in separate trials. There was a significant nonlinear relationship between probe RTT and test stream rate, with a jump from $<10ms$ to $>300ms$ when the stream rate went above 350 packets per second. Figure 5 shows probability density estimates for probe RTTs for different test stream packet rates.

There was a statistically significant difference between the means and variances of all three distributions, according to *p*-value tests. In our testbed, an adversary would be able to determine if the packets in a test stream were reaching the control plane by comparing the distribution of probe RTTs before sending the test stream with the distribution of probe RTTs while sending the test stream.

Learning about Forwarding Tables An adversary could leverage the ability to determine whether a test stream's packets go to the controller to learn about the rules installed in the switch's forwarding tables. For example, in our testbed, the adversary can learn that there are forwarding rules installed to direct traffic from 1.1.1.2 to 1.1.2 but not 1.2 or 1.3 by measuring probe RTTs while sending one test packet, and then, in separate trials, measuring probe RTTs while sending test packet streams with templates: `mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:02`, `mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:03`, and `mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:00`.

Figure 6 shows the distributions of the RTTs for each test packet stream. The test packet stream to h_2 did not cause a statistically significant shift in the RTT distribution, indicating that the switch must have handled the packets without forwarding them to the controller. However, the test packet streams to h_3 and h_4 caused a statistically significant shift in the distribution, indicating that the switch did not match a rule and ended up in the control plane.

What estimated probability densities using Kernel Density Estimation

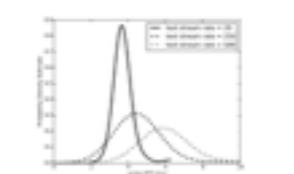


Figure 5: Probe RTT distribution estimates with test streams of different rates that get processed by the control plane.

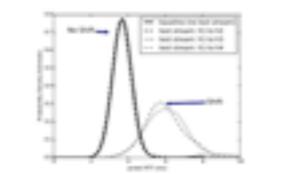


Figure 6: Probe RTT distribution estimates when the adversary sends test streams with no matching forwarding rules.

packets did not match a rule and ended up in the control plane.

Once an adversary knows that a forwarding rule exists in the network, they can then figure out which fields of the rule are wildcarded by measuring baseline probe RTTs while sending a test packet stream with a template that matches the rule; then in a separate trial for each packet header field, setting the value of the field to be random in the packet stream's template. If the rule has a field wildcarded, randomizing it in the packet stream will not change the RTT statistics. If the rule does not have a field wildcarded, randomizing it in the packet stream will cause more packets to go to the controller, shifting the RTT statistics.

Table 2 shows probe RTT statistics while randomizing different fields of test stream templates that match forwarding rules on the switch in our testbed. The probe RTT and variance remain low when the `mac_source` field is randomized, which indicates that the rule forwarding traffic to 00:00:00:00:00:02 must have its `mac_source` field wildcarded. Randomizing any other field in either source results in a higher probe RTT and variance, which indicates that the forwarding rule must require exact matches in those fields.

Example Case Attack Planning If a network's forwarding rules match on both the source and destination, an adversary can build a communication graph for the hosts in the network by checking for rules that forward packets between each pair of hosts. Hosts that have many edges in the communication graph (i.e., communicate with many other hosts) may be critical to the network and an ideal target for DoS attacks, which hosts with few edges may be less importantly used and an ideal target for the adversary to infect while minimizing disruption to the network.

Stream Template	Rate	probe RTT (average)	probe RTT (variance)
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:01	100	0.84	0.06
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:02	100	0.87	0.02
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:03	100	10.82	0.32

Table 2: Probe RTT statistics while sending test streams with different packet header fields randomized.

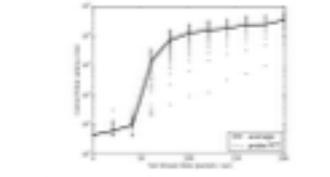


Figure 7: Probe RTT at test stream packet rate rates, for a test stream where each packet invokes a rule installation.

3.2 Is the controller installing forwarding rules?

To determine the effect of the controller installing a forwarding rule on the switch, we measured probe RTTs while sending a sequence of two test packet streams that cause the MAC learner on the controller to install one rule for each packet in the second test stream. More specifically, we generated two streams of MAC addresses: R_1 and R_2 , each of which had 1000 unique random MAC addresses. We first sent a stream with the template: `mac_source=R_1, mac_dest=R_2`. The switch forwarded each packet of the stream to the controller, which added the rule, `mac_source=R_1, mac_dest=R_2`, each of which had 1000 unique random MAC addresses. We then sent a stream with the template: `mac_source=R_2, mac_dest=R_1`. The switch also forwarded each of these packets to the controller. However, since the controller knew the location of each destination MAC address (due to the first stream), it installed a forwarding rule on the switch for these packets with that destination MAC.

Figure 7 shows the RTT of the timing probes during the second test stream, as the stream rate varied. Each packet in this stream caused the controller to install a forwarding rule. Since slow test streams greatly increased the probe RTT, a test stream that caused all flow rules per second to get installed increased the probe RTT by approximately 100ms, for example. Figure 8 shows, one stream at each low rate did not affect the probe RTT if the controller was processing packets in the test stream but not installing rules.

On our testbed, an adversary would be able to easily determine whether the control plane installed new forwarding rules in response to a stream of packets that they sent.

Learning about controller logic If an adversary can determine whether the controller installs rules in response to a test packet stream, they can learn about the controller's logic. For example, they could determine if the controller can install a rule that forwards traffic from

Stream Template	Rate	RTT (average)	RTT (variance)
Total 1			
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:01	100	0.89	1.07
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:02	100	0.82	1.08
Total 2			
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:01	100	0.11	1.73
mac_source=00:00:00:00:00:01, mac_dest=00:00:00:00:00:02	100	0.12	1.08

Table 3: Probe RTT statistics during one trial to determine the response of packets that causes a rule to be installed. High RTT followed by low RTT indicates that rules are installed.

a MAC entry by performing trials that use different sequences of packet streams. Table 3 shows results for each trial; the adversary sends two low rate test streams, and then one high rate test stream. The latency for the third stage of the first trial is significantly higher, indicating that the first two streams did not install rules. In the second trial, however, the latency during the second stage was much higher, suggesting that the controller may have installed rules. The third stage of the trial confirms this, as the RTT statistics indicate low controller load, even though the first stream rate is high. Thus, the adversary can infer that when the controller observes a packet from a $x \rightarrow y$ followed by a packet from $y \rightarrow x$, it will install a rule forwarding $y \rightarrow x$.

Example Case DoS Attacks OpenFlow switches store forwarding rules in TCAM memory that allows them to match packets quickly. TCAM memory is expensive and power hungry, so modern OpenFlow switches can only store up to approximately 10,000 forwarding rules in TCAM. Any additional rules are put into tables in standard memory, and matching packets against these rules is slower. If an adversary learns the sequence of packets that causes the controller to install a rule, they can saturate TCAM flow tables with a small number of packets and greatly degrade the network performance of all flows.

4. CONCLUSION

With control plane inference attacks, adversaries can learn about a SDN network's configuration and functionality without compromising the network's infrastructure. We developed an inference attack that uses the RTT of probes that travel through the control plane as an indicator of its load. Our attack allows an adversary to learn more about a network, compared with previous attacks. We have demonstrated that our inference attack is effective on real OpenFlow hardware and provided examples of how it can help adversaries stage other more advanced attacks. An SDN adoption process, if it is important to develop defenses for this class of attacks.

5. REFERENCES

- [1] See <http://www.openflow.org/wiki/>, its network <http://www.openflow.org/wiki/Network-Topology> and our <http://www.openflow.org/wiki/Network-Topology>.
- [2] Ryu, <http://github.com/ryuno>.
- [3] B. Xiao, V. Kuznetsov, and P. Yang, OpenFlow: A security analysis. In *Network Protocols (ICNP)*, 2012 (pp. 822-832). International Conference on, page 14, IEEE, 2013.
- [4] J. Yang, Y. Zhou, J. Zhang, and C. Xu, An inference attack model for flow table overflow in software-defined network. In *Proc. of the 10th ACM SIGCOMM*, 2013.

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



Potential Defenses

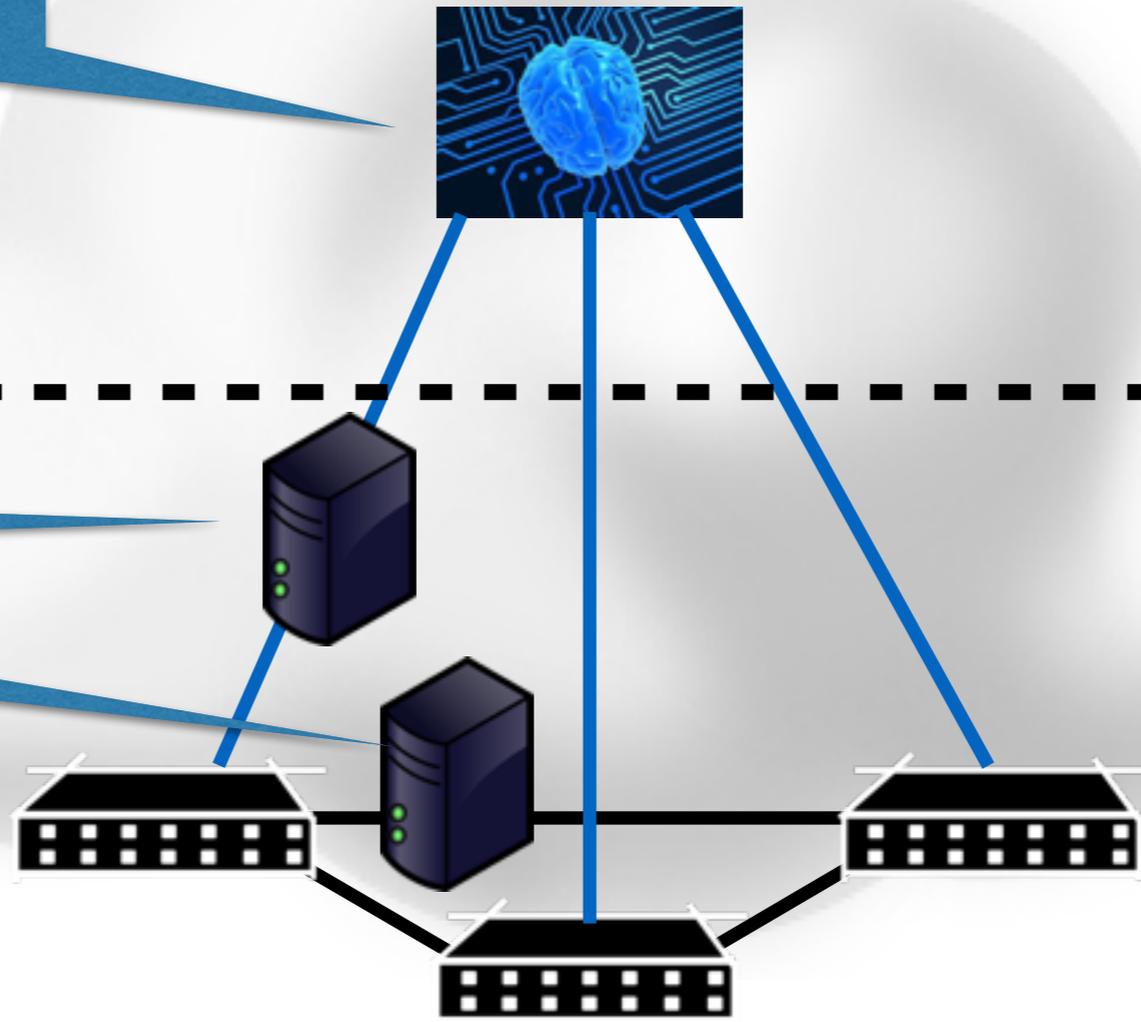


Control Plane

Real time controllers?

Middlebox detectors and filters?

Programmable switches? (OFX, P4)



Data Plane