COMSE-6998

DDoS Attack Detection & Mitigation in SDN

FINAL VIVA PRESENTATION 2014-12-08

Presented by Chao CHEN (cc3736)

Key Words

DDoS Attack **Detection** and **Mitigation**

Type: **ICMP** Flood
SYN Flood
DNS Amplification
UDP Flood

InMon **sFlow-RT** + Floodlight controller + Mininet

SDN **Application** to perform DDoS Protection

RESEARCH BACKGROUND

SCHEME DESIGN

APPLICATION DEVELOPMENT

ENVIRONMENT ESTABLISHMENT

TEST & EVALUATION

RESEARCH BACKGROUND

Research Background

Real Time detection and mitigation with **lowest cost** of device **deployment**

sFlow

Research Background

sFlow = **sampled** Flow

SDN analytics and control using sFlow standard

Device Capability → Easy Deployment

Physical Device:	Cisco Nexus 3000/3100 series
	IBM c/g/m/r/s/x/y series
	Juniper EX 2200/3200/3300/4200/6200 series
	……
Virtual Device:	**OpenVSwitch**
	Apache
	Nginx
	……
sFlow Collectors:	**InMon sFlow-RT**
	Brocade Network Advisor
	……

Research Background

sFlow + Openflow

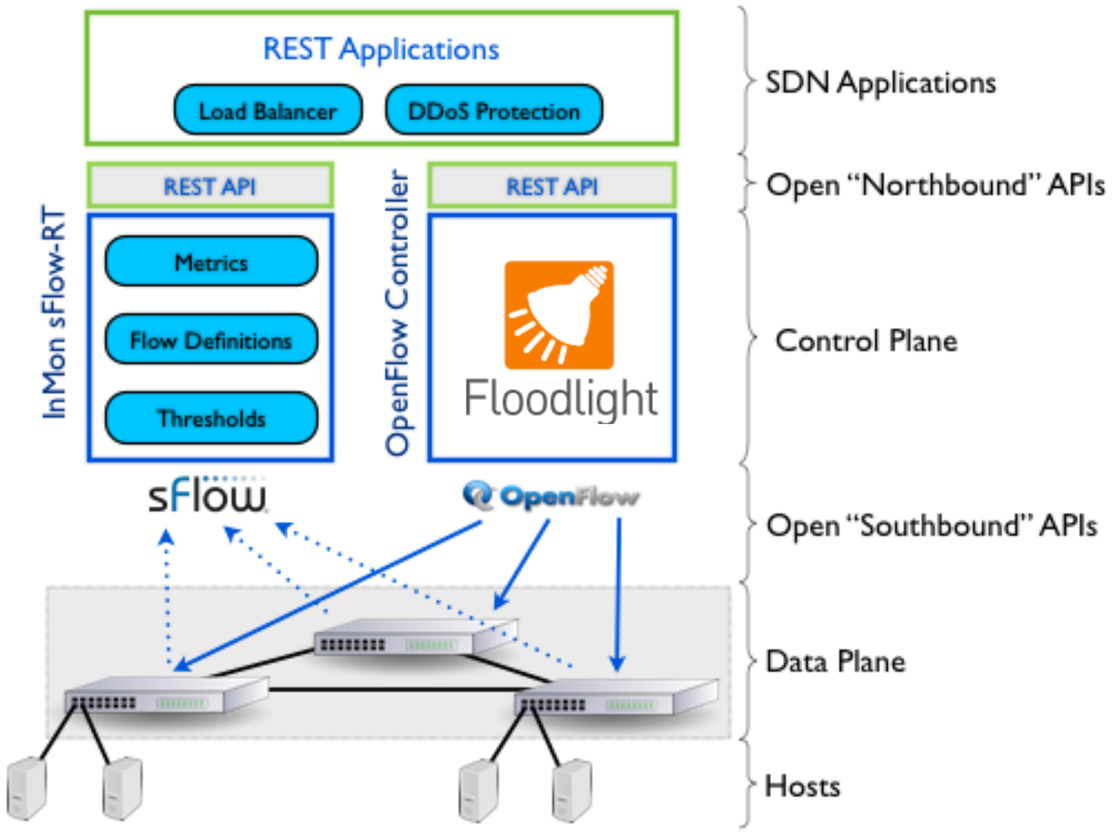
1. switch samples packets

2. switch sends the header of sampled packets to sFlow-RT

3. sFlow-RT maps it into fine-grained flow(e.g. tcpflags=SYN, icmptype=3…)

4. if exceed the threshold, trigger an event

5. events accessible from external apps through **REST API**



Research Background

sFlow + Openflow

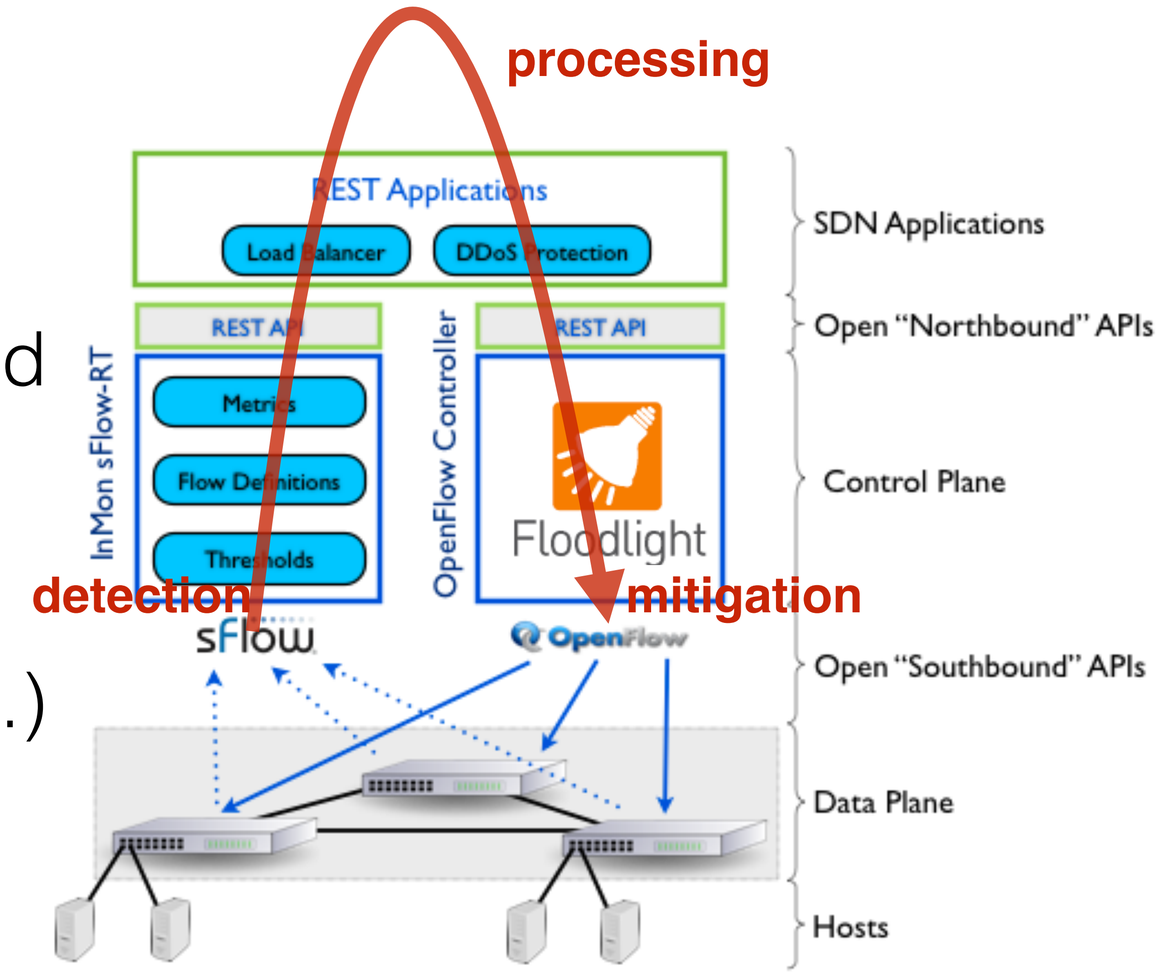
1. switch samples packets

2. switch sends the header of sampled packets to sFlow-RT

3. sFlow-RT maps it into fine-grained flow(e.g. tcpflags=SYN, icmptype=3…)

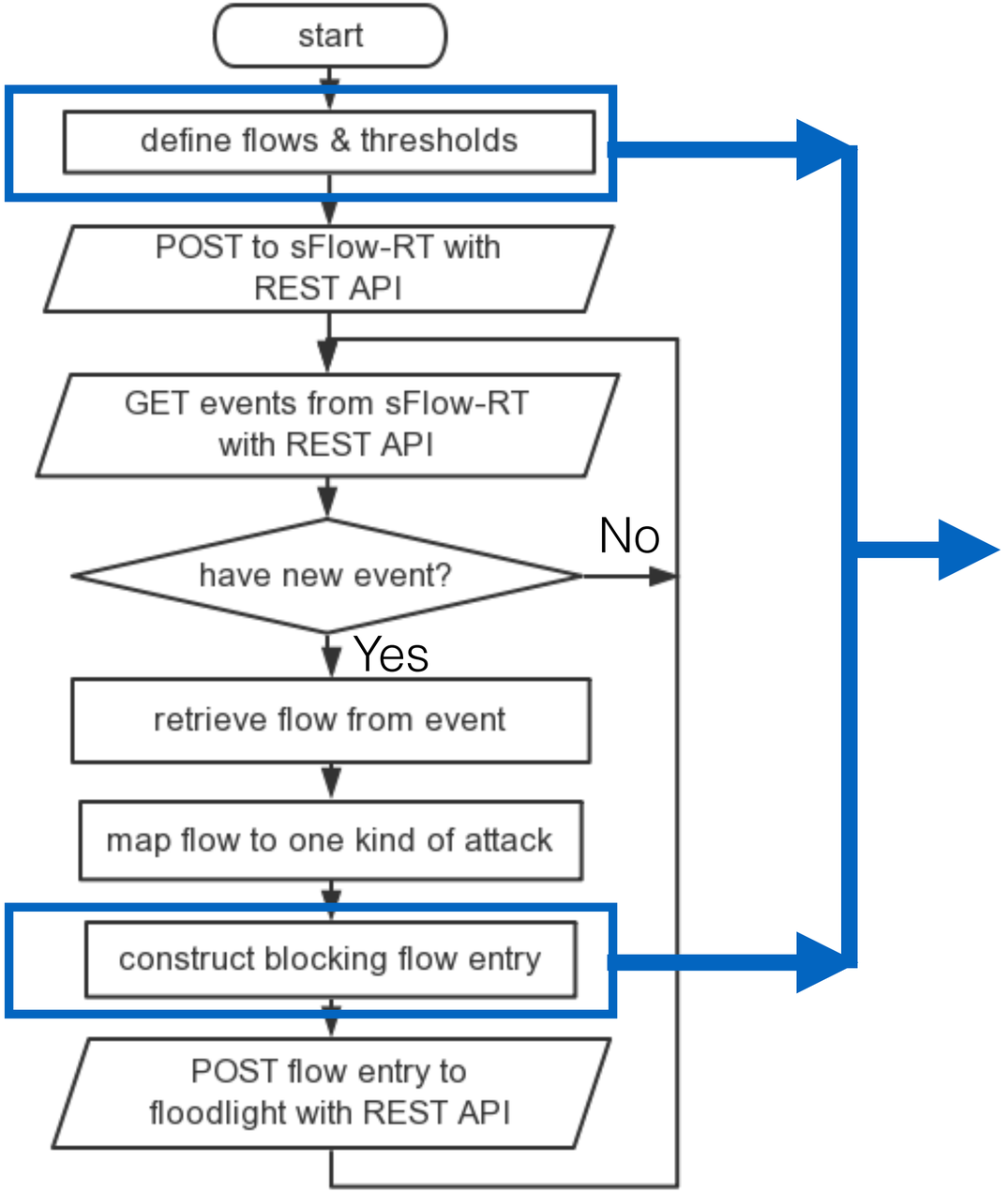
4. if exceed the threshold, trigger an event

5. events accessible from external apps through REST API



SCHEME DESIGN

Overall Flowchart of Application



need to be specified for different kinds of attacks

ICMP Flood Attack

Mechanism:

Each device in the botnet ping the server at a high rate.

Flow Definition:

ipsource=0.0.0.0/0,
ipdestination=10.0.0.2/32, #suppose h2 is the server
outputifindex!=discard, #packet is not discarded
ipprotocol=1 #ICMP

Match Field in blocking flow entry:

ether-type, protocol, src-ip, dst-ip

SYN Flood Attack

Mechanism:

Each device in the botnet sends TCP SYN packets to the server at a high rate.

Flow Definition:

ipsource=0.0.0.0/0,
ipdestination=10.0.0.2/32, #suppose h2 is the server
outputifindex!=discard, #packet is not discarded
tcpflags~…….1.=1 #TCP SYN packet

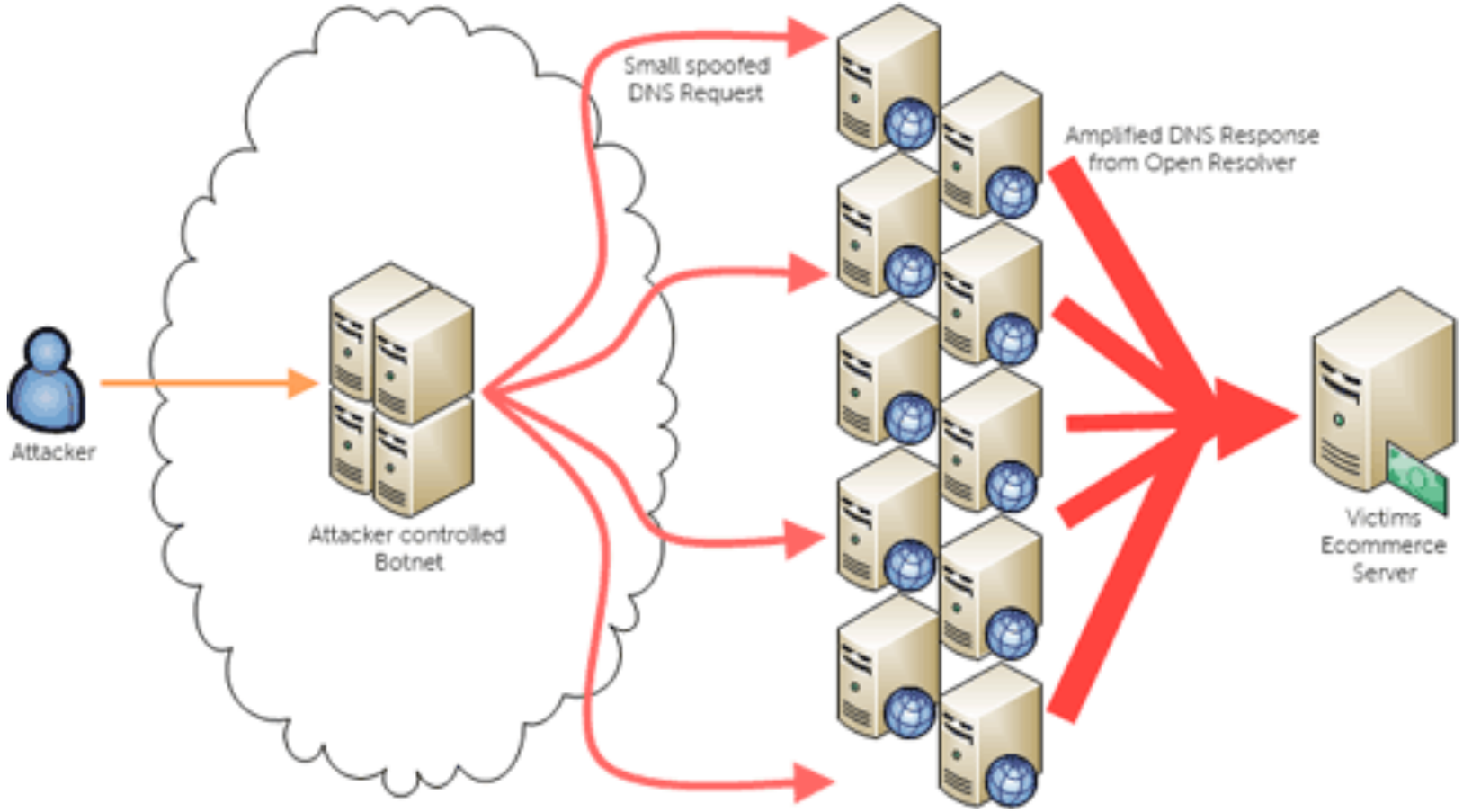
Match Field in blocking flow entry:

ether-type, protocol, src-ip, dst-ip

DNS Amplification Attack

Mechanism:

Each device in the botnet sends DNS query to several DNS servers with src-ip=victim's ip. (take ANY(15) for example)



DNS Amplification Attack

Protect at the DNS servers (instead of the victim)

Flow Definition:

ipsource=0.0.0.0/0,
ipdestination=[10.0.0.1/32, 10.0.0.2/32], #suppose h1 and h2 are the DNS servers
outputifindex!=discard, #packet is not discarded
dnsqr=false,
dnsqtype=255

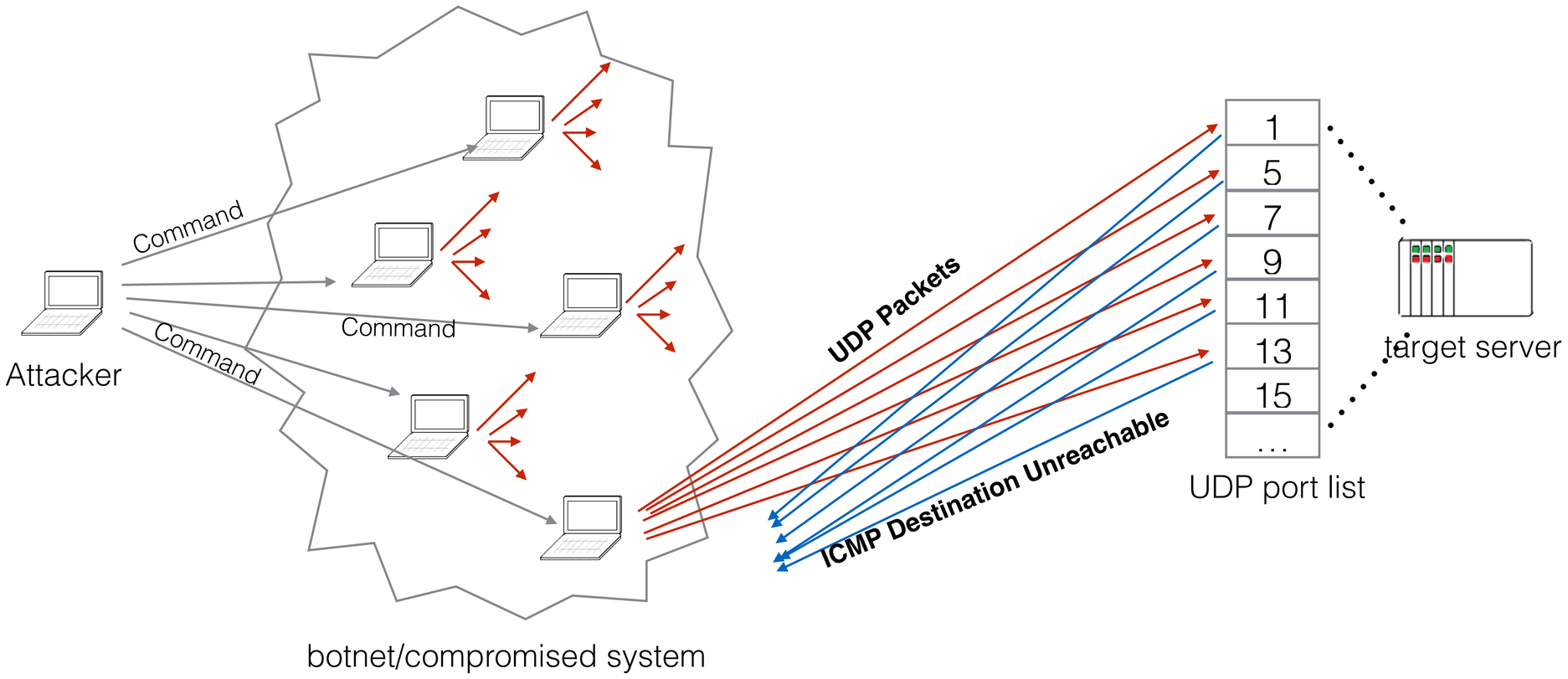
Match Field in blocking flow entry:

ether-type, protocol, src-ip, dst-ip

UDP Flood Attack

Mechanism:

Each device in the botnet sends UDP packets to all the ports if the server



UDP Flood Attack

Protect by monitoring **ICMP Destination Unreachable** packets

Flow Definition:

ipsource=**10.0.0.2/32, #reversed**
ipdestination=**0.0.0.0/0,**
outputifindex!=discard, #packet is not discarded
ipprotocol=1, #ICMP
icmptype=3, #Destination Unreachable

Match Field in blocking flow entry:

ether-type, protocol, **src-ip=dst-ip_in_flow**, **dst-ip=server-ip**

APPLICATION DEVELOPMENT

Application Development

python

Import **requests** & **json** to perform GET/PUT/POST via REST API

Different attacks are implemented similarly.

Take ICMP Flood attack as example.

Definition of flows, thresholds,…:

```
# define ICMP flood attack attributes #
icmp_flood_keys = 'inputifindex,ethernetprotocol,macsource,macdestination,ipprotocol,ipsource,ipdestination'
icmp_flood_metric_name = 'icmp_flood'
icmp_flood_threshold_value = 100
icmp_flood_filter = 'group:ipsource:lf=external&group:ipdestination:lf=internal&outputifindex!=discard&ipprotocol=1'
icmp_flood_flows = {'keys': icmp_flood_keys, 'value': value, 'filter': icmp_flood_filter}
icmp_flood_threshold = {'metric': icmp_flood_metric_name, 'value': icmp_flood_threshold_value}
```

POST the definition to sFlow-RT:

```
# define flows and threshold of ICMP flood
r = requests.put(sFlow_RT + '/flow/' + icmp_flood_metric_name + '/json', data=json.dumps(icmp_flood_flows))
r = requests.put(sFlow_RT + '/threshold/' + icmp_flood_metric_name + '/json', data=json.dumps(icmp_flood_threshold))
```

Attack classification & Static Flow Entry Push:

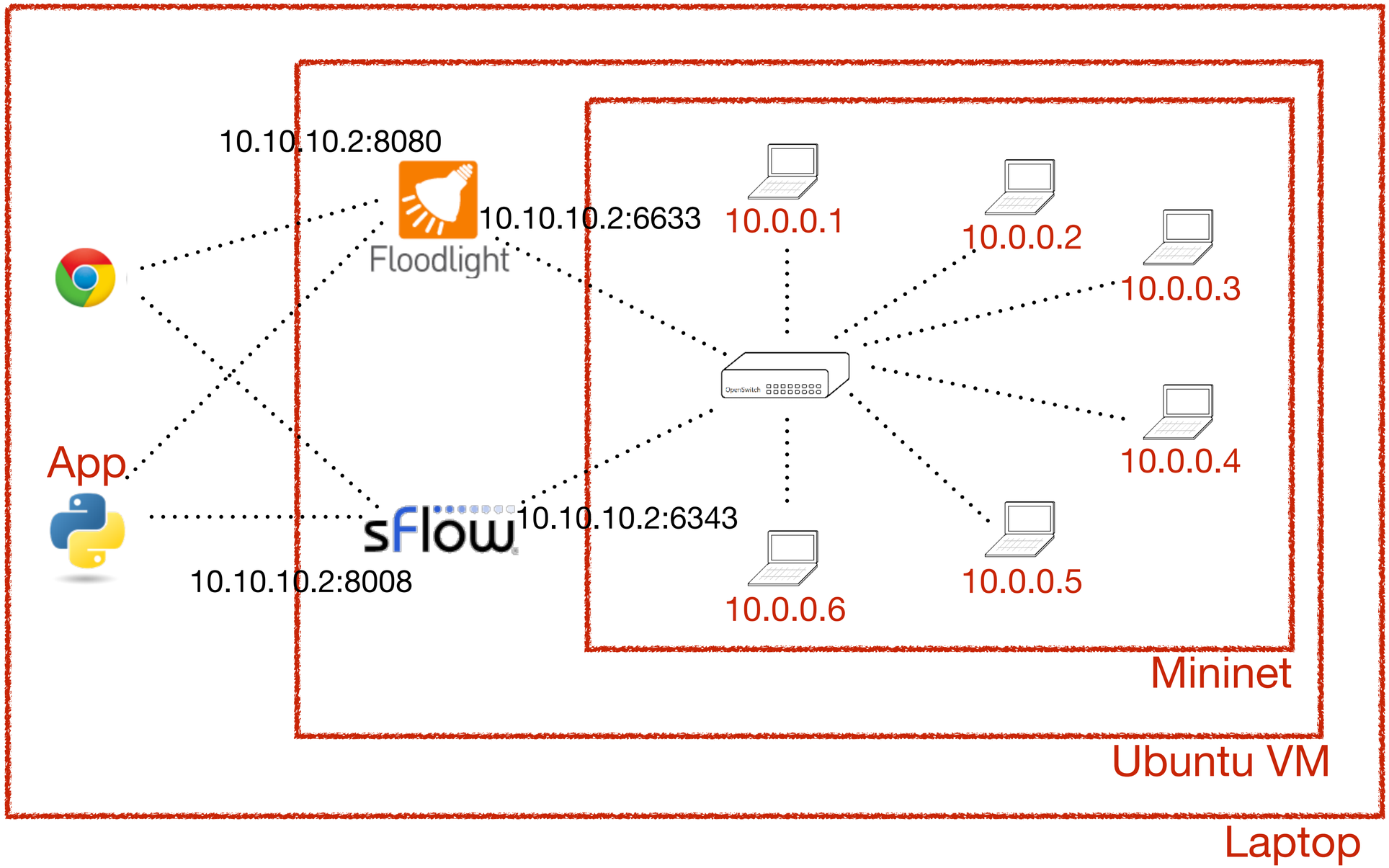
```
elif e['metric'] == icmp_flood_metric_name:
    r = requests.get(sFlow_RT + '/metric/' + e['agent'] + '/' + e['dataSource'] + '.' + e['metric'] + '/json')
    metrics = r.json()
    if metrics and metrics.__len__() > 0:
        metric = metrics[0]
        if metric.__contains__("metricValue") \
                and metric['metricValue'] > icmp_flood_threshold_value\
                and metric['topKeys']\
                and metric['topKeys'].__len__() > 0:

            for topKey in metric['topKeys']:
                if topKey['value'] > icmp_flood_threshold_value:
                    key = topKey['key']
                    print key,
                    parts = key.split(',')

                    message = {'switch': 1,
                               'name': 'ICMP_block_'+parts[5],
                               'ether-type': parts[1],
                               'protocol': parts[4],
                               'src-ip': parts[5],
                               'dst-ip': parts[6],
                               'priority': fw_priority,
                               'active': 'true'}
                    push_data = json.dumps(message)
                    r = requests.post(floodlight + '/wm/staticflowentrypusher/json', data=push_data)
                    black_list.append([time.time()+block_time, push_data])
                    result = r.json()
                    print ""
                    print result['status']
            print ""
```

ENVIRONMENT ESTABLISHMENT

Environment Establishment



TEST & EVALUATION

Test & Evaluation

Launch floodlight: ./floodlight.sh

```
mininet@mininet-vm:~$ cd floodlight/
mininet@mininet-vm:~/floodlight$ ./floodlight.sh
Starting floodlight server ...
INFO [net.floodlightcontroller.core.module.FloodlightModuleLoader:main] Loading default modules
INFO [net.floodlightcontroller.core.internal.Controller:main] Controller role set to null
```

Launch InMon sFlow-RT: ./start.sh

```
mininet@mininet-vm:~/sflow-rt$ sudo ./start.sh
2014-12-08T09:40:15-0800 INFO: Listening, sFlow port 6343
2014-12-08T09:40:15-0800 INFO: Listening, http://localhost:8008
2014-12-08T09:40:15-0800 INFO: init.js started
2014-12-08T09:40:15-0800 INFO: init.js stopped
```

Launch InMon sFlow-RT: sudo ./topo.sh

```
mininet@mininet-vm:~$ sudo ./topo.sh
*** Creating nodes
*** Creating links
*** Configuring hosts
h1 h2 h3 h4 h5 h6
*** Starting network
*** Running CLI
*** Starting CLI:
mininet>
```

set s1 is a sFlow agent, and set up bridge between s1 and sFlow-RT

Node: s1 (root)

```
root@mininet-vm:~# sudo ovs-vsctl -- --id=@sflow create sflow agent=eth0 target=
\"127.0.0.1:6343\" sampling=10 polling=20 -- -- set bridge s1 sflow=@sflow
```

ICMP Flood Attack

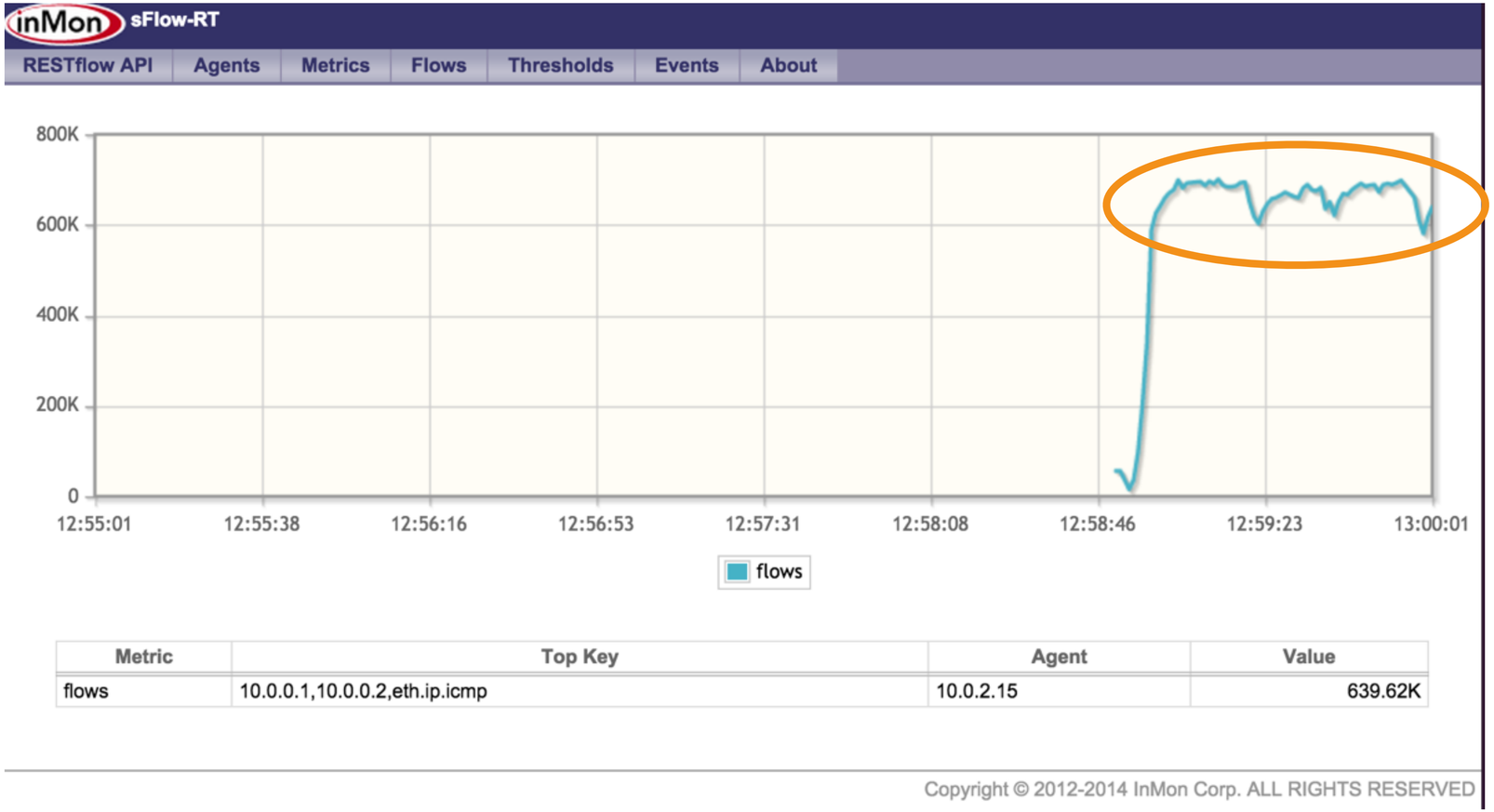
Without mitigation:

h1 ICMP attack on h2 with: *ping -f 10.0.0.2*

Node: h1

```
root@mininet-vm:~# ping -f 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
.
```

network traffic flow



Metric	Top Key	Agent	Value
flows	10.0.0.1,10.0.0.2,eth.ip.icmp	10.0.2.15	639.62K

attack from h4

ICMP Flood Attack

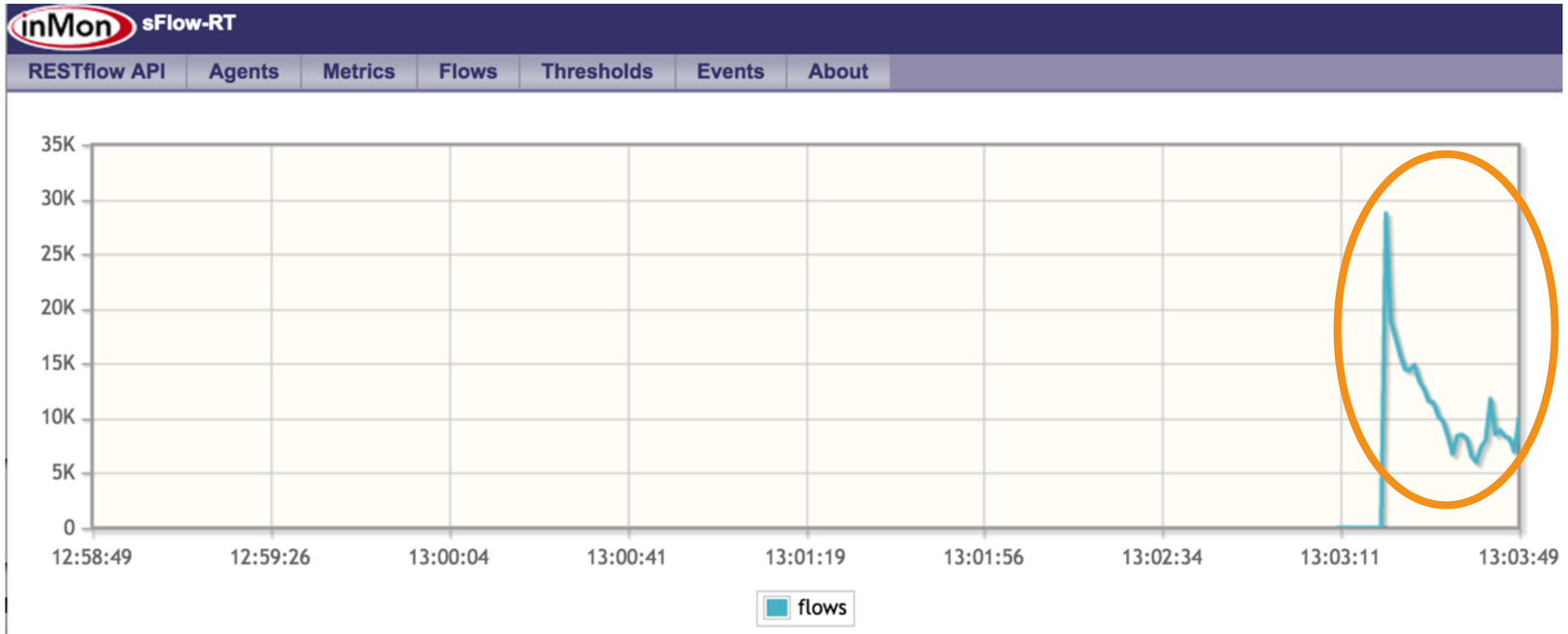
With mitigation:

h4 ICMP attack on h2

Node: h4

```
root@mininet-vm:~# ping -f 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
.
.
```

network traffic flow



attack from h4 is mitigated

Metric	Top Key	Agent	Value
flows	10.0.0.4,10.0.0.2,eth.ip.icmp	10.0.2.15	10.05K

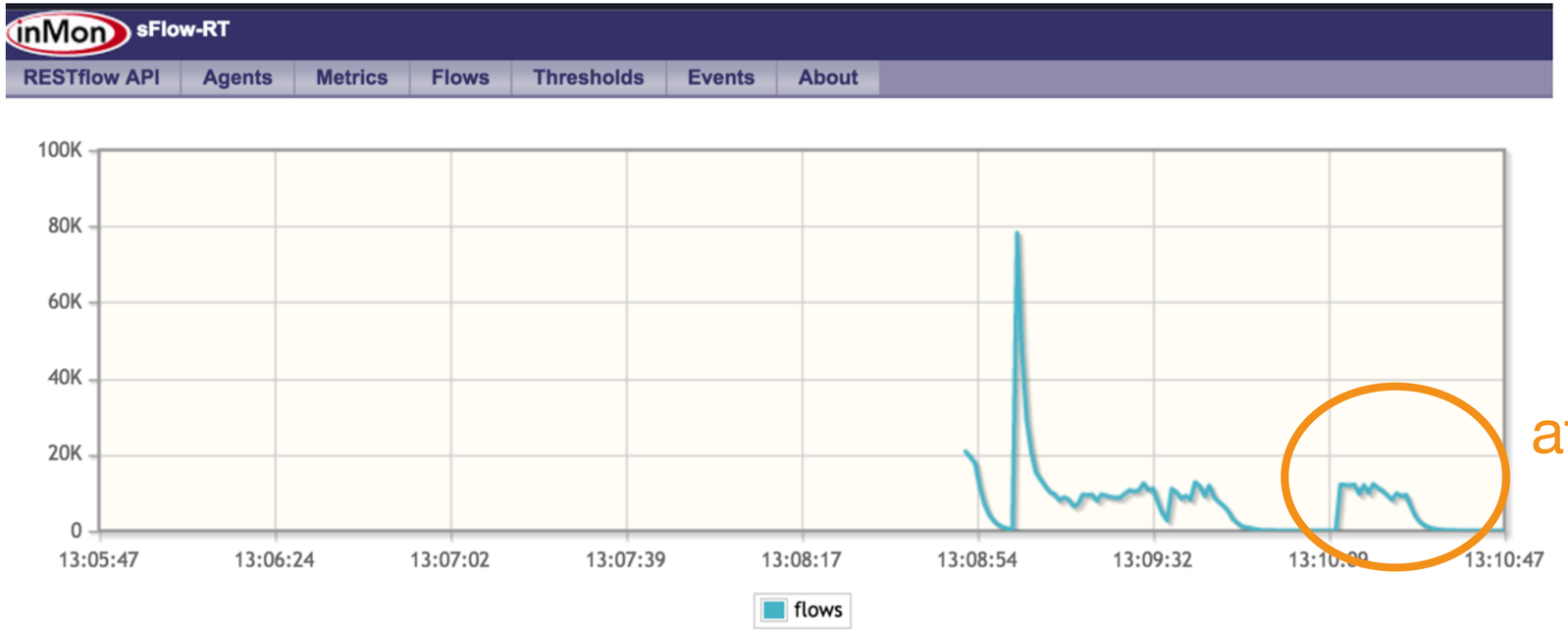
ICMP Flood Attack

Continue: h5 ICMP attack on h2

Node: h5

```
root@mininet-vm:~# ping -f 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
.
.
```

network traffic flow



attack from h5 is mitigated

Metric	Top Key	Agent	Value
flows	10.0.0.5,10.0.0.2,eth.ip.icmp	10.0.2.15	0.44

Test & Evaluation

ICMP Flood Attack

‘subflows’ in ICMP Attack Flow

inMon sFlow-RT

RESTflow API | Agents | Metrics | Flows | Thresholds | Events | About

inputifindex	ethernetprotocol	macsource	macdestination	ipprotocol	ipsource	ipdestination	bytes
115	2048	0A0055555555	0A0022222222	1	10.0.0.5	10.0.0.2	0.000
111	2048	0A0033333333	0A0022222222	1	10.0.0.3	10.0.0.2	0.000
113	2048	0A0044444444	0A0022222222	1	10.0.0.4	10.0.0.2	0.000
117	2048	0A0066666666	0A0022222222	1	10.0.0.6	10.0.0.2	0.000

Events triggered in this case

inMon sFlow-RT

RESTflow API | Agents | Metrics | Flows | Thresholds | Events | About

ID	Time	Name	Metric	Threshold	Value	Agent	Data Source
3	Mon Dec 08 2014 13:10:13 GMT-0500 (EST)	icmp_flood	icmp_flood	100	504.76	10.0.2.15	115
2	Mon Dec 08 2014 13:09:35 GMT-0500 (EST)	icmp_flood	icmp_flood	100	504.76	10.0.2.15	111
1	Mon Dec 08 2014 13:09:02 GMT-0500 (EST)	icmp_flood	icmp_flood	100	504.76	10.0.2.15	113
0	Mon Dec 08 2014 13:08:48 GMT-0500 (EST)	icmp_flood	icmp_flood	100	504.76	10.0.2.15	117

Flowtable of s1 (attacked by h3, h4, h6)

Flows (4)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
45035996311043704	32767	ethertype=0x0800, proto=1, src=10.0.0.3, dest=10.0.0.2		1019	99862	20 s	0 s
45035996311043710	32767	ethertype=0x0800, proto=1, src=10.0.0.6, dest=10.0.0.2		486	47628	67 s	0 s
45035996311043704	32767	ethertype=0x0800, proto=1, src=10.0.0.4, dest=10.0.0.2		2453	240394	53 s	0 s
45035996311043704	32767	ethertype=0x0800, proto=1, src=10.0.0.4, dest=10.0.0.2		2453	240394	162 s	0 s

SYN Flood Attack

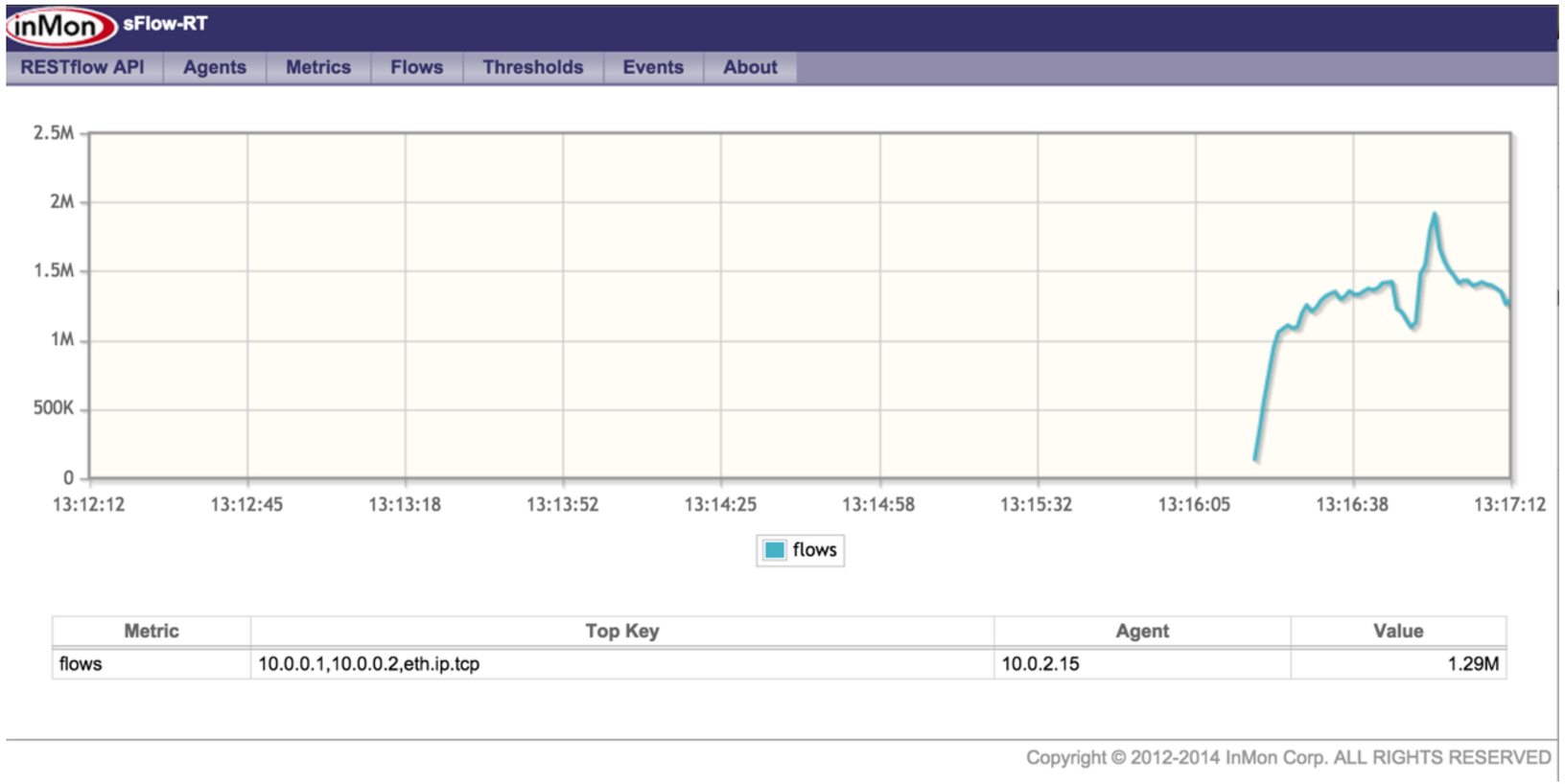
Without mitigation:

h1 SYN attack on h2 with: *ping —tcp -p 80 —flag syn -rate 2000 —count 20000000 —no-capture —quiet 10.0.0.2*

Node: h1

```
root@mininet-vm:~# sudo nping --tcp -p 80 --flags syn -rate 2000 --count 2000000
0 --no-capture --quiet 10.0.0.2
```

network traffic flow



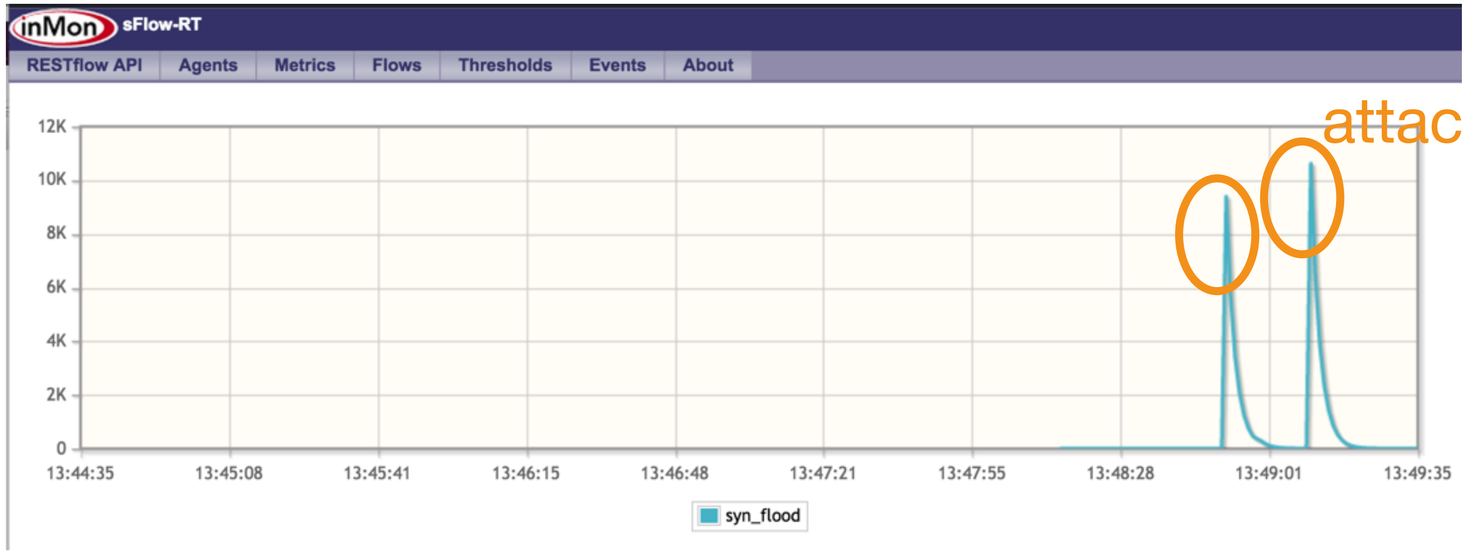
Metric	Top Key	Agent	Value
flows	10.0.0.1,10.0.0.2,eth.ip.tcp	10.0.2.15	1.29M

SYN Flood Attack

With mitigation:

h6 and h4 SYN attack on h2

SYN Flood Traffic



attacks from h6 and h4 are mitigated

Flowtable of s1 (attacked by h3, h4, h5, h6)

Cookie	Priority	Match	Action	Packets	Bytes	Age	Timeout
45035997250776930	32767	ethertype=0x0800, proto=6, src=10.0.0.3, dest=10.0.0.2		5450928	294350112	173 s	0 s
45035997250776930	32767	ethertype=0x0800, proto=6, src=10.0.0.6, dest=10.0.0.2		1985156	107198424	87 s	0 s
45035997250776930	32767	ethertype=0x0800, proto=6, src=10.0.0.4, dest=10.0.0.2		317690	17155260	35 s	0 s
45035997250776930	32767	ethertype=0x0800, proto=6, src=10.0.0.3, dest=10.0.0.2		8183826	441926604	382 s	0 s
45035997250776930	32767	ethertype=0x0800, proto=6, src=10.0.0.5, dest=10.0.0.2		1561825	84338550	53 s	0 s

Test & Evaluation

DNS Amplification Attack & UDP Flood Attack:

Cannot simulate attacks → No test result yet

Test & Evaluation

Future Work:

1. Test on DNS Amplification Attack & UDP Flood Attack
2. {new_sample_rate, new_threshold}
   =update(old_sample_rate, old_threshold, network_congestion, server_status,…)
3. Sample Theory is efficient on large flows.
   Think about {tiny flows x n}
4. Reasonable unblock mechanism

Q&A