NDSS SYMPOSIUM/2020

Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches

Menghao Zhang[1], Guanyu Li[1], Shicheng Wang[1], Chang Liu[1], Ang Chen[2], **Hongxin Hu**[3], Guofei Gu[4], Qi Li[1], Mingwei Xu[1], Jianping Wu[1]

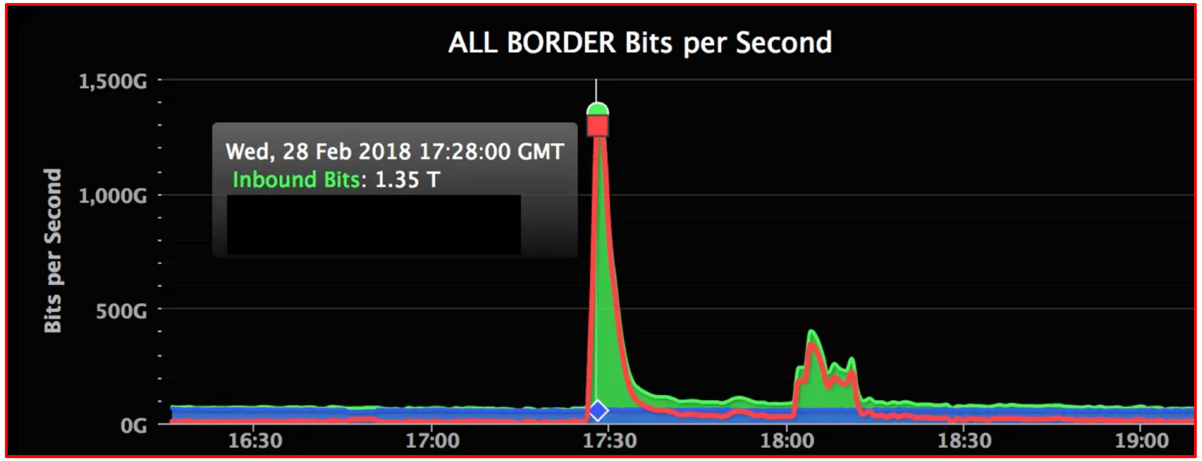
1 清華大學 Tsinghua University 2 RICE 3 CLEMSON UNIVERSITY 4 TEXAS A&M UNIVERSITY

DDoS Attacks are Getting Worse

The Rise of IoT Botnet Threats and DDoS attacks

Corero, 2018

Increase in scale



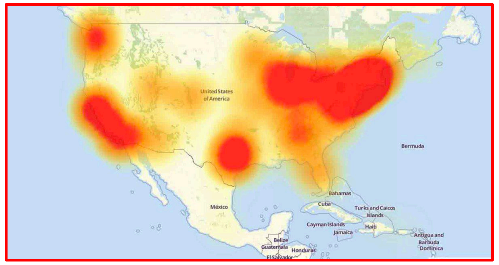
The GitHub Blog, 2018

Increase in diversity

The latest DDoS attacks are mostly multi-vector and morph over time

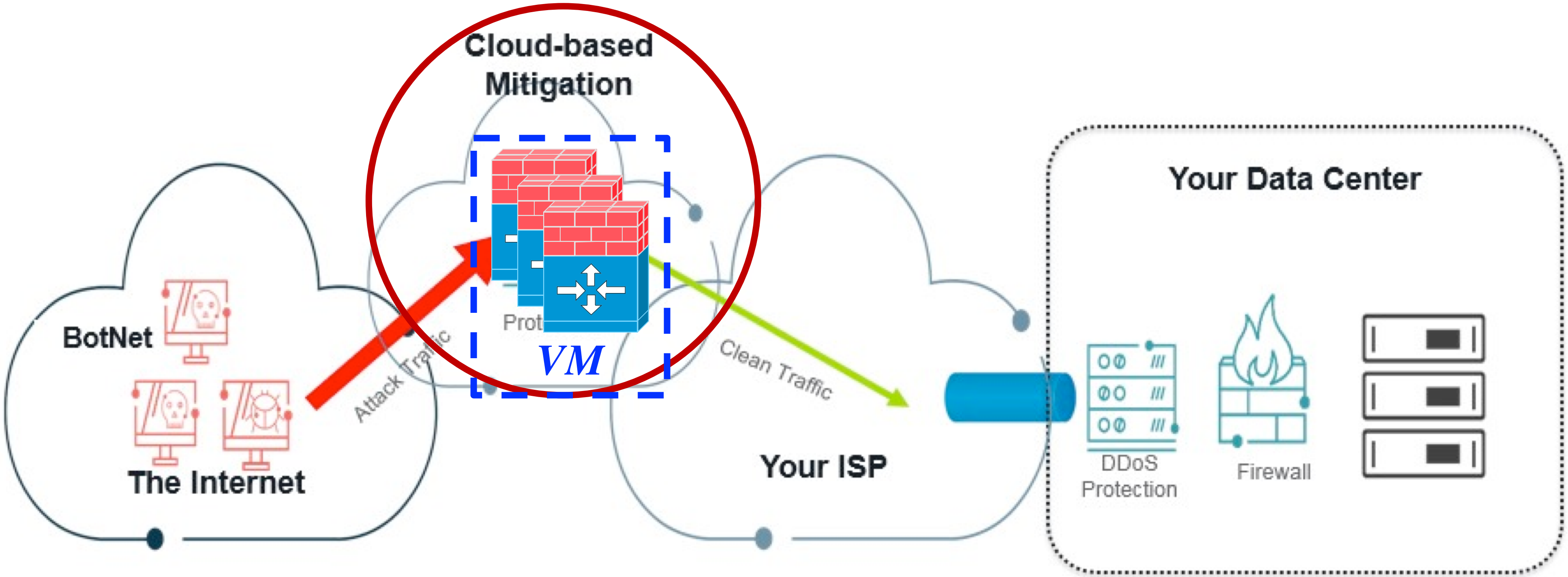
Help Net Security, 2019

Major DDoS attack on Dyn disrupts AWS, Twitter, Spotify and more



Datacenter Dynamics, 2016

DDoS Defense Today – Traffic Scrubbing Center



Middlebox

- *High performance*
- *Expensive, inflexible*

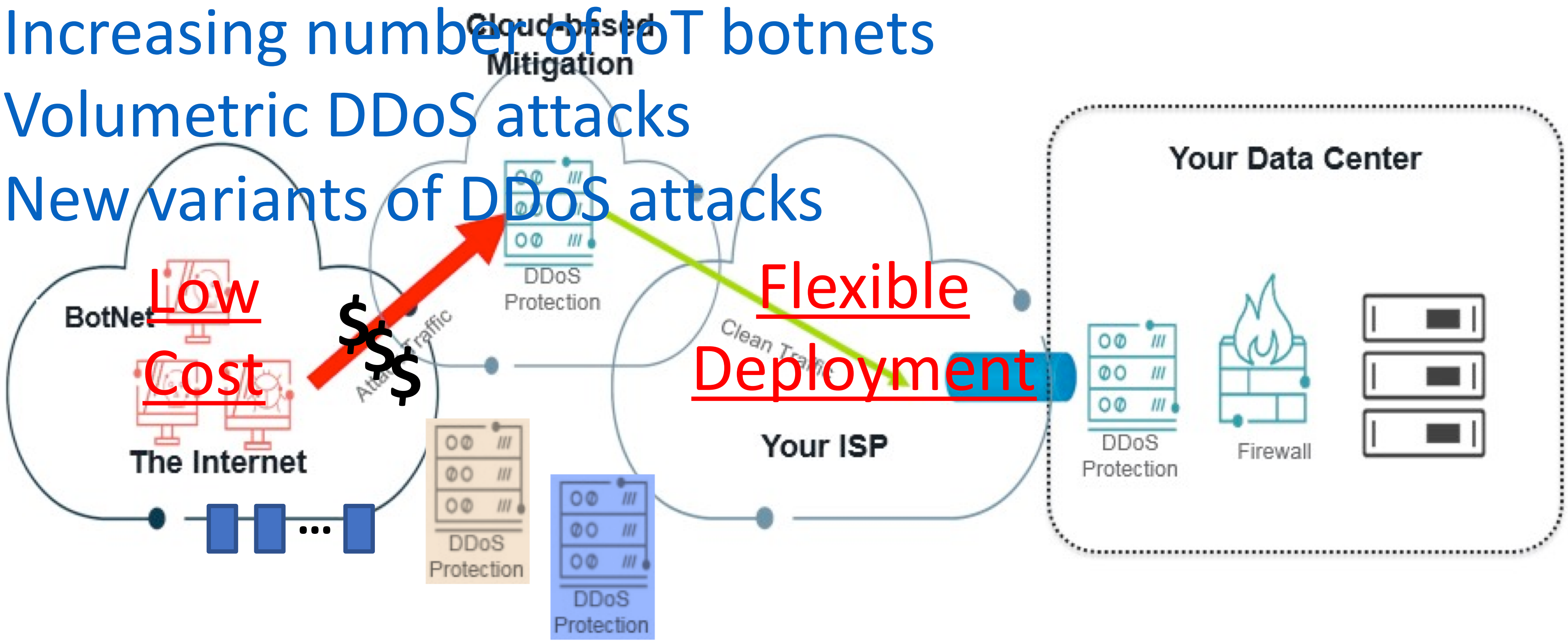
Network Function Virtualization

- *Flexible, elastic*
- *Low Performance*

*malware.news

Ideal DDoS Traffic Scrubbing Service

- Increasing number of IoT botnets
- Volumetric DDoS attacks
- New variants of DDoS attacks



Low Cost

Flexible Deployment

High Performance

New Opportunities: Programmable Switches

Parser Program

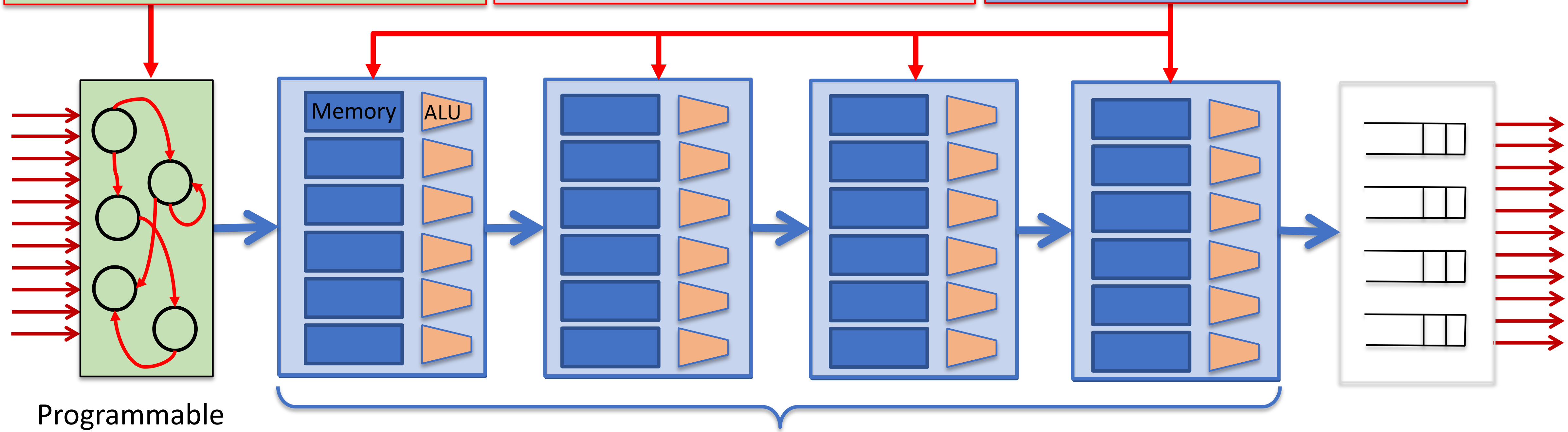
```
parser parse_ethernet {
   extract(ethernet);
   return switch(ethernet.ethertype) {
      0x8100 : parse_vlan_tag;
      0x0800 : parse_ipv4;
      0x8847 : parse_mpls;
      default: ingress;
}
```

Header and Data Declarations

```
header_type   ethernet_t      { … }
header_type   l2_metadata_t { … }

header     ethernet_t     ethernet;
header     vlan_tag_t
vlan_tag[2];
metadata   l2_metadata_t l2_meta;
```

Tables and Control Flow

```
table port_table { … }

control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```



Programmable Parser

Programmable Match-Action Pipeline

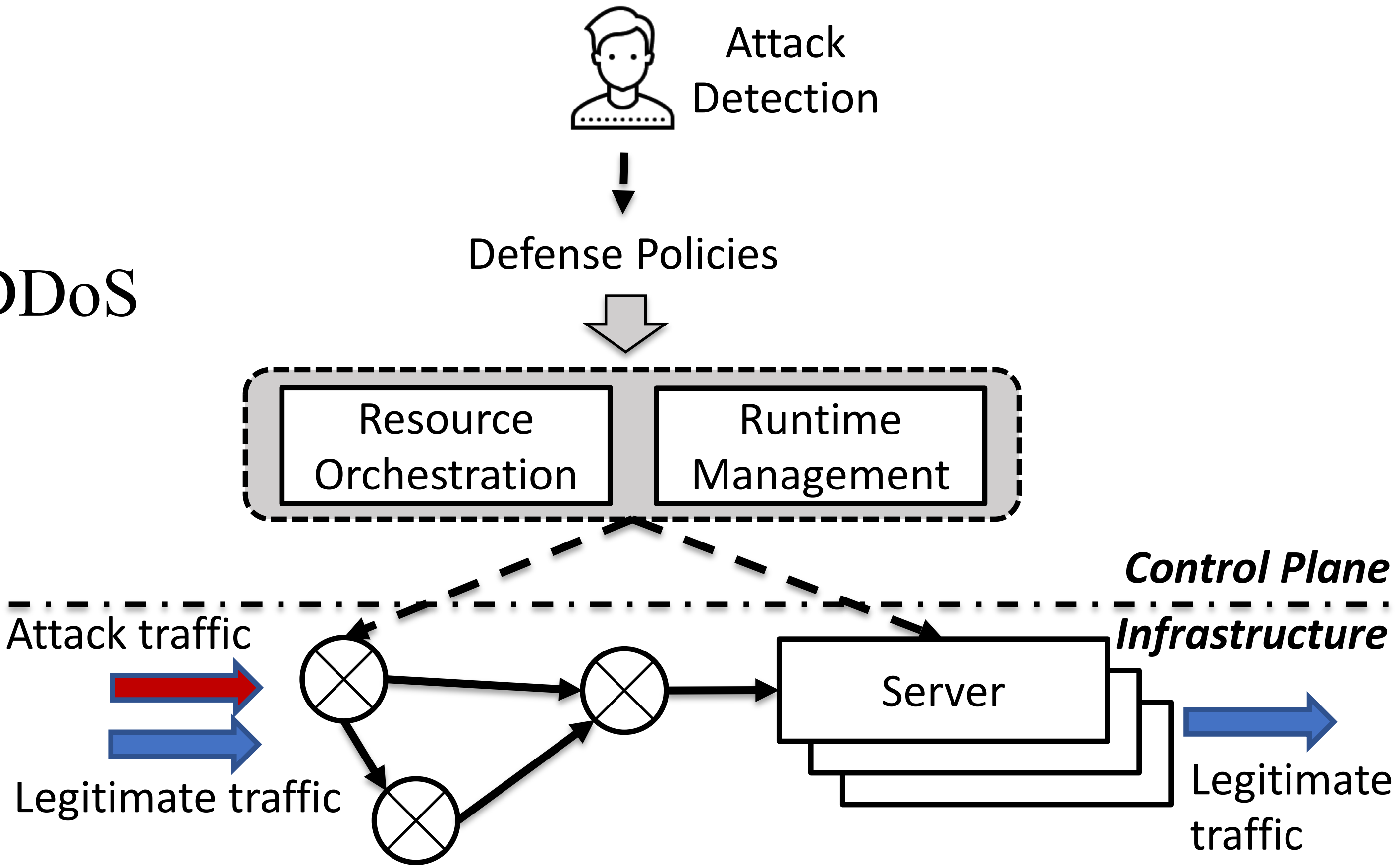
New Opportunities: Programmable Switches

- *Programmed using P4*
  - ***Flexibility** to support future defenses*
- *Same power and cost as fixed-function switches*
  - ***Lower** unit capital **cost***
- *Programs always run at line-rate*
  - ***High** packet processing **performance***

Poseidon: Bring these benefits to DDoS defense

Poseidon System Overview

- Deployment scenario
  - Traffic scrubbing center
- Threat model
  - Volumetric and Dynamic DDoS attacks against victims
- Workflow
  - Attack detection
  - Policy declaration
  - Attack mitigation



Poseidon Design Challenges

- Policy representation
  - Accommodate to heterogeneous DDoS defense mechanisms
- Resource orchestration
  - Limited on-chip resources and restrictive computational models in switching ASICs
- Handling dynamic attacks
  - Naively recompile the P4 program for switches
    - State loss and flow interruption
  - Update the defenses when all flow states are no longer needed
    - Waste of precious and high-density defense resources (i.e., switching ASICs)

1. Expressing Defense Policies

- Observation
  - Key components common to many volumetric attacks
- Adapted from NetCore [POPL'12]
  - High modularity
  - High-level abstractions and customizations for DDoS defense

Expression

$E \quad ::= \quad \mathsf{v} \mid \mathsf{h} \mid M(\vec{\mathsf{v}}) \mid E \diamond E$

Predicate

$P \quad ::= \quad E \circ E \mid P \& P \mid P|P \mid \neg P$

Monitor

$M \quad ::= \quad \mathsf{count}(P, \vec{\mathsf{h}}, \mathsf{every}) \mid \mathsf{aggr}(P, \vec{\mathsf{h}}, \mathsf{every})$

Action

$A \quad ::= \quad \mathsf{drop} \mid \mathsf{pass} \mid \mathsf{log} \mid \mathsf{rlimit} \mid \mathsf{sproxy} \mid \mathsf{puzzle}$

Policy

$C \quad ::= \quad A \mid \mathsf{if}\ P\!: C\ \mathsf{else}: C \mid (C|C)$

Details in our paper!

1. Policy Example

- SYN Flood Defense

```
syn_count = count(pkt.tcp.flag == SYN, [ip.src], 5)
ack_count = count(pkt.tcp.flag == ACK, [ip.src], 5)

if syn_count([pkt.ip.src]) - ack_count([pkt.ip.src]) > T:
    drop
else if syn_count([pkt.ip.src]) == ack_count([pkt.ip.src]):
    pass
else:
    sproxy
```

***POSEIDON**: 9 lines of code*

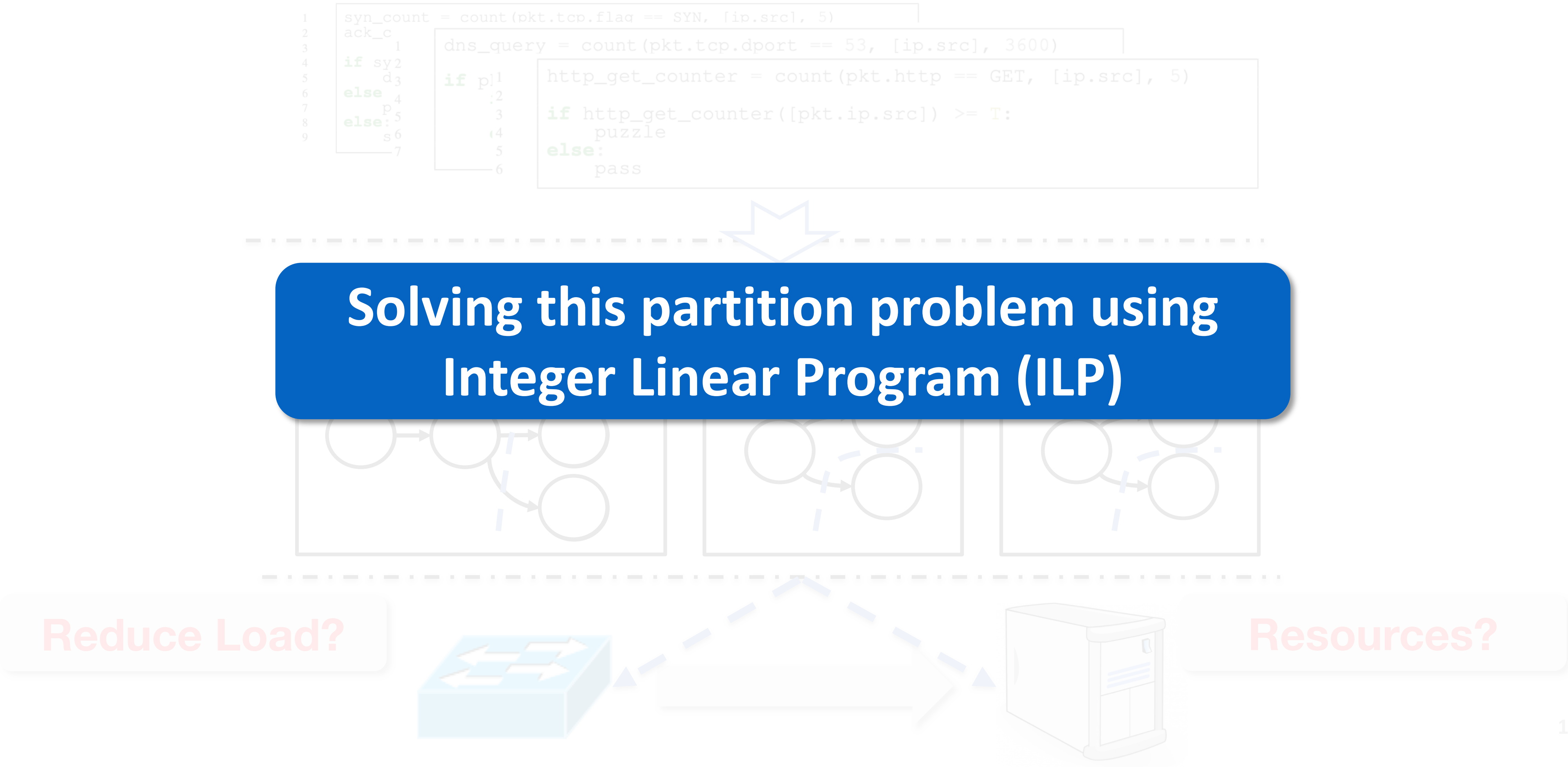
```
/* Header declaration */
struct headers {
    ether_t ether;
    ipv4_t  ipv4;
    tcp_t   tcp;
}
// Definitions of ether_t, ipv4_t and tcp_t are omitted

/* Metadata declaration */
header_type syn_proxy_meta_t {
    fields { ... }
}
metadata syn_proxy_meta_t meta;
// We remove the specific fields of metadata

/* Parser declaration */
parser parse_ether {
    extract(ether);
    return select(latest.etherType) {
        ETHERTYPE_IPV4: parse_ipv4;
        default: ingresss;
    }
}
parser parse_ipv4 {
    extract(ipv4);
    return select(latest.protocol) {
        IP_PROTOCOLS_TCP : parse_tcp;
        default: ingress;


}
par


}
// Calculation of checksum is ignored

/* Monitor (counter) declaration */
register syn_count_cm_sketch_row1 {
    width : WIDTH;
    instance_count : COLUMN;
}
register syn_count_cm_sketch_row1_last_period {
    width : WIDTH;
    instance_count : COLUMN;
}
register ack_count_cm_sketch_row1 {
    width : WIDTH;
    instance_count : COLUMN;
}
register ack_count_cm_sketch_row1_last_period {
    width : WIDTH;
    instance_count : COLUMN;
}
// We omit the other rows of two count-min sketches

/* Match-Action Table declaration */
table syn_count_update_table {
    read {
        tcp.syn : exact;
    }
```

***P4**: 91 lines of code*

2. Analyzing Defense Primitives

Primitives	**Switch Component**	**Switch Resource Usage**	**Server Component**
monitors			
count($P, \vec{h}$, every)	match-action entry + count-min sketch	stages: 2, hash functions: $\lceil \log_{1/2} \delta \rceil$, stateful ALUs: 6, SRAM: for the ϕ biggest elements in a set, in order to achieve a relative error bound of ε with probability δ, usage = $\frac{64\lceil \log_{1/2} \delta \rceil}{\varepsilon \phi}$	N/A
aggr($P, \vec{h}$, every)	match-action entry + count-min sketch	stages: 2, hash functions: $\lceil \log_{1/2} \delta \rceil$, stateful ALUs: 6, SRAM: for the ϕ biggest elements in a set, in order to achieve a relative error bound of ε with probability δ, usage = $\frac{64\lceil \log_{1/2} \delta \rceil}{\varepsilon \phi}$	N/A
actions			
drop	flow entry	stages: 1, hash functions: 0, stateful ALUs: 0, SRAM: negligible	N/A
pass	flow entry	stages: 1, hash functions: 0, stateful ALUs: 0, SRAM: negligible	N/A
rlimit	meter + flow entry	stages: 3, hash functions: 1, stateful ALUs: 0, SRAM: in order to achieve a false positive rate of ε, usage = $\frac{8n}{ln(1/(1-\varepsilon))}$	N/A
sproxy	handshake proxy + session relay	stages: 3, hash functions: 2, stateful ALUs: 4, SRAM: in order to achieve a false positive rate of ε, usage = $\frac{32n}{ln(1/(1-\varepsilon))}$	N/A
puzzle	-	-	CAPTCHA
log	selecting, grouping	stages: 3, hash functions: 2, stateful ALUs: 2, SRAM: in order to achieve a false positive rate of ε, usage = $\frac{32n}{ln(1/(1-\varepsilon))}$	aggregation
branches			
if . . . else . . .	tag-based match action	stages: 1, hash functions: 0, stateful ALUs: 0, SRAM: negligible	N/A

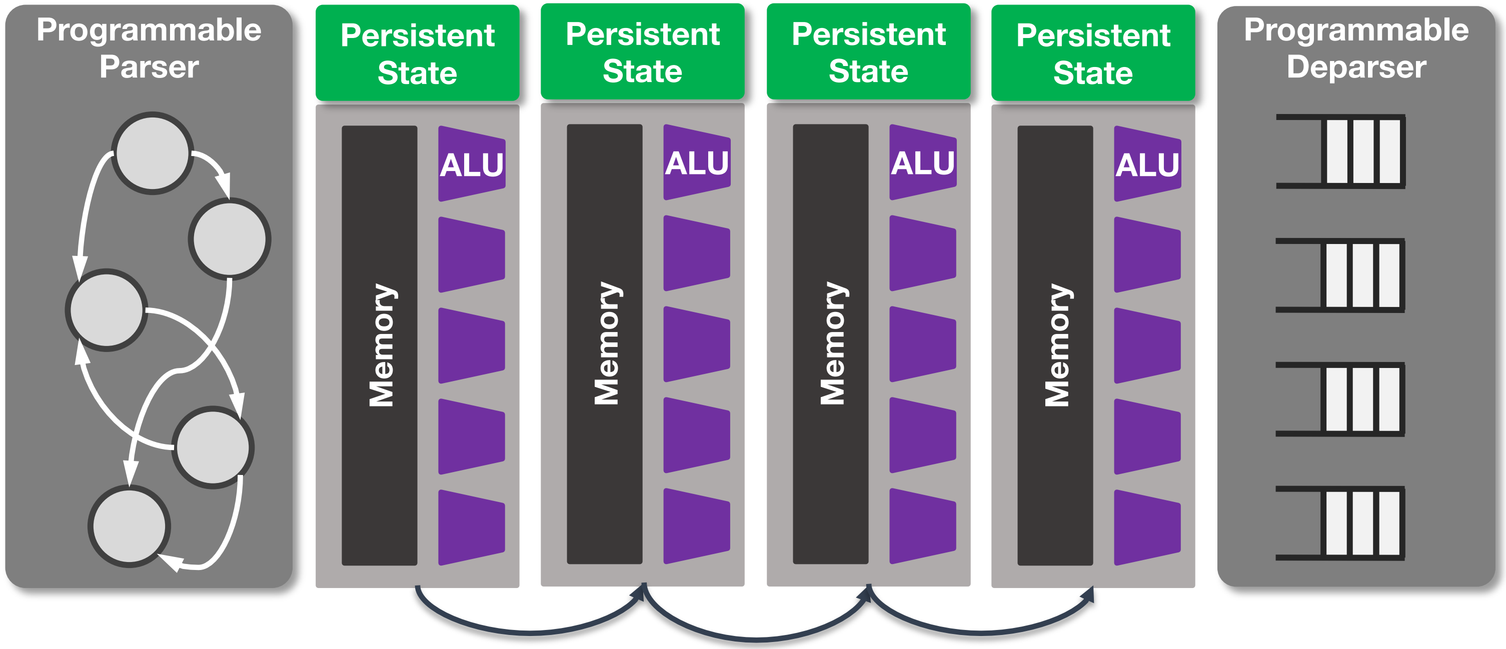
2. Placing Defense Primitives



Solving this partition problem using Integer Linear Program (ILP)

2. Partition ILP

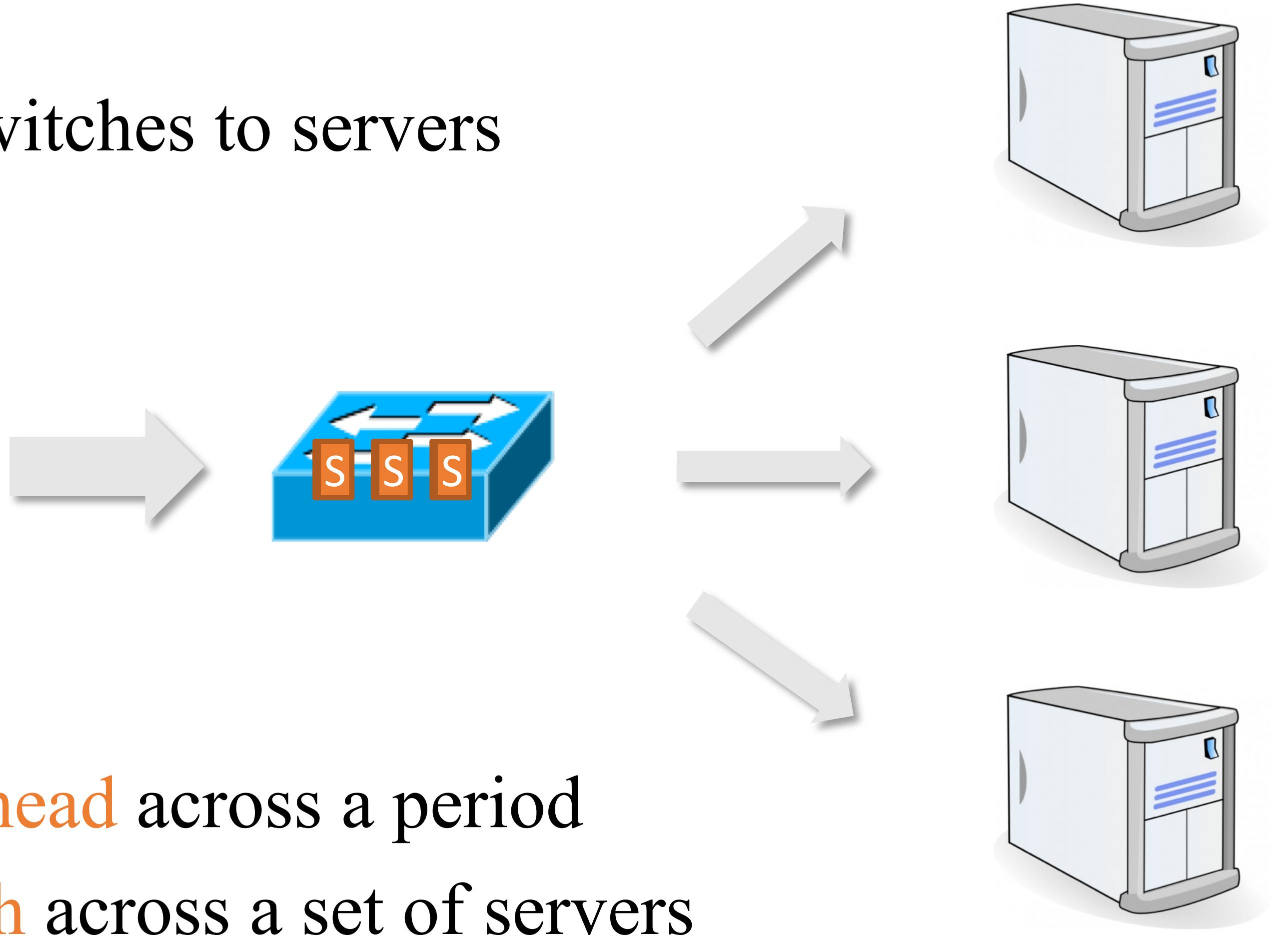
Constraints
Stateful Memory
Number of Actions
Total Stages
Node Order



Goal: **Minimize packets sending to servers**

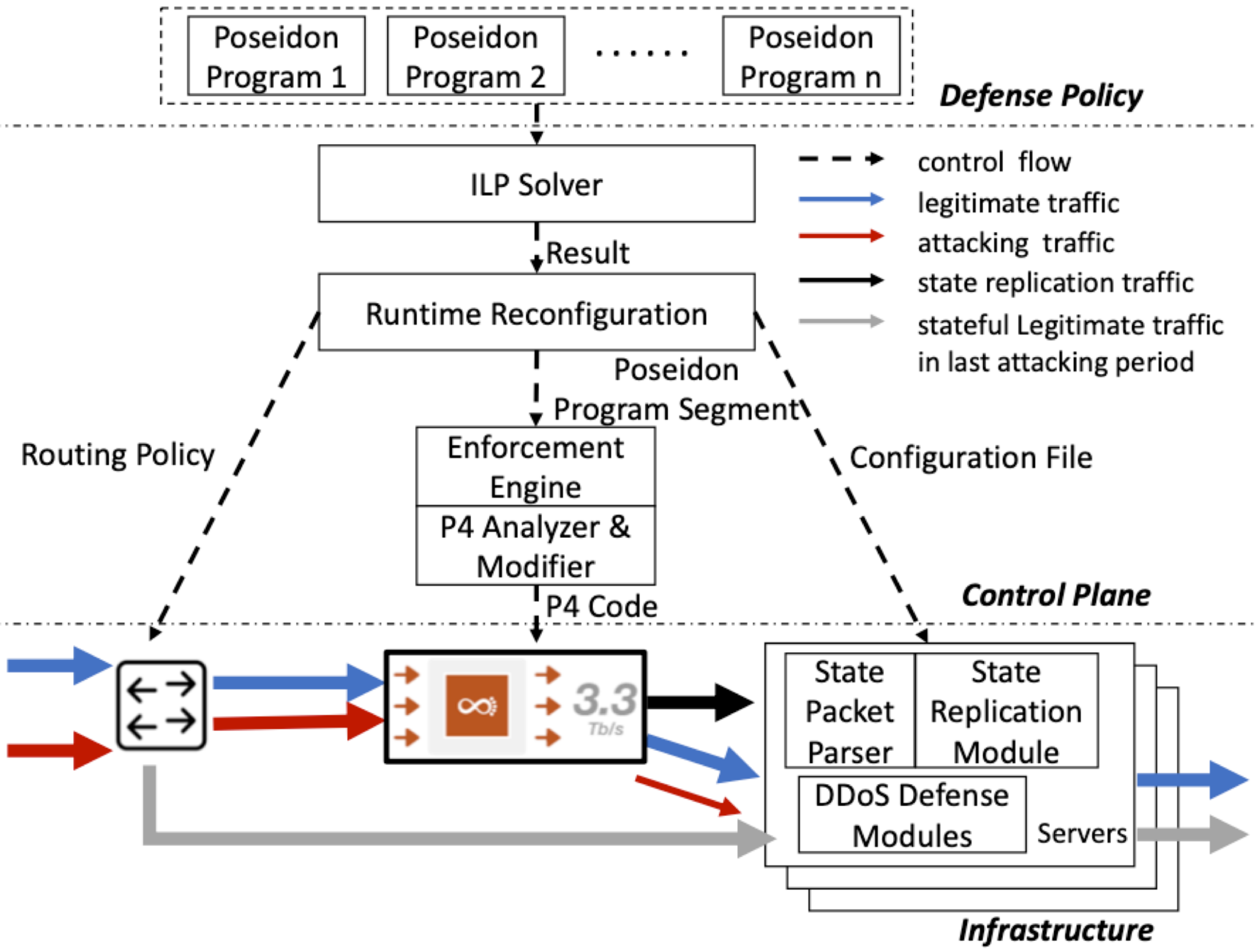
3. Handling Dynamic Attacks

- Key idea
  - Copy necessary states in the switches to servers
- States requiring replication
  - Identify the states which will still take effect for legitimate traffic even when attacks finish
- Approach to replication
  - Distribute the replication overhead across a period
  - Spread the traffic from a switch across a set of servers

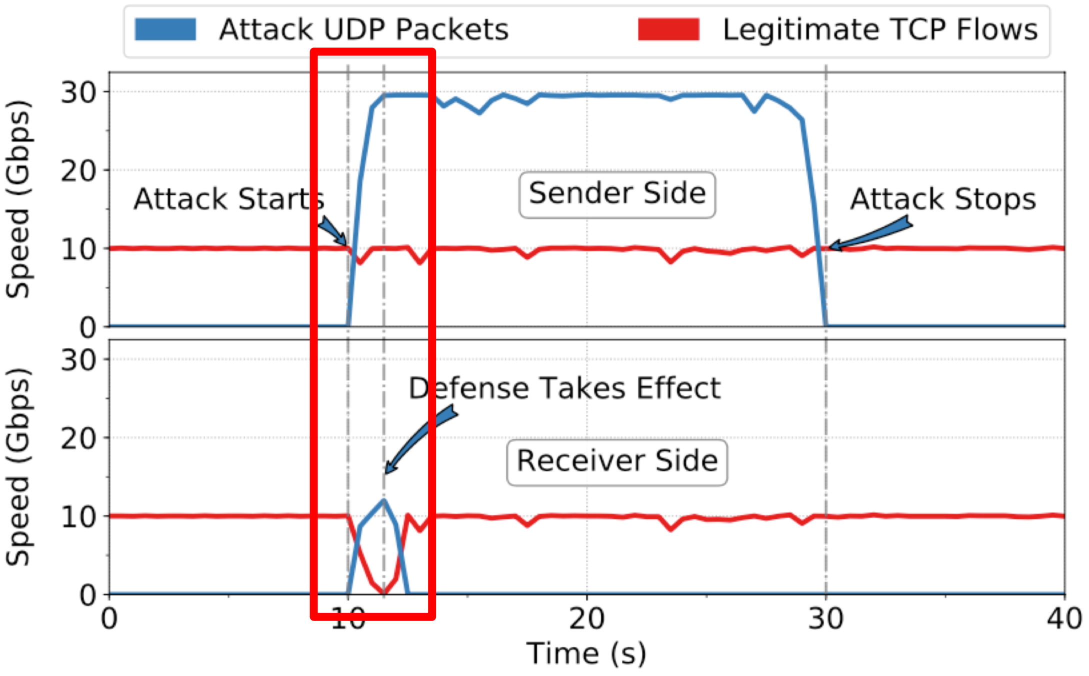


Implementation & Evaluation

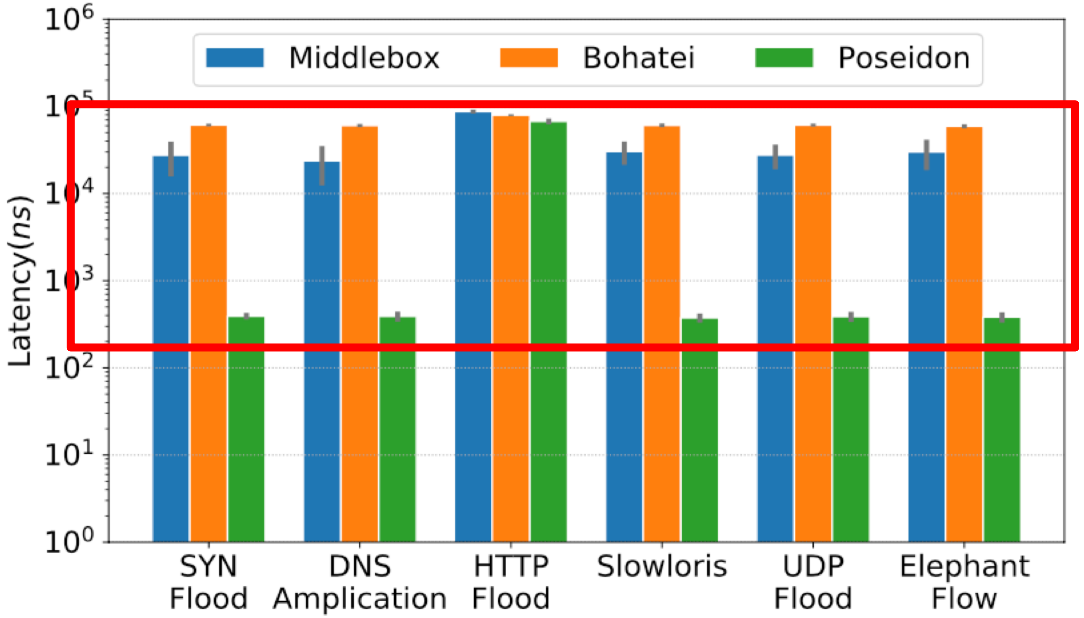
- Implementation
  - Policy primitives
    - P4 for switch part
    - DPDK for server part
  - Resource orchestration
    - Policy enforcement engine
  - Runtime management
    - Switch/server interface
    - State replication mechanism
- Evaluation
  - Real-world testbeds + Trace-driven evaluations



Overall Effectiveness



Throughput restoration for legitimate flows during attacks



End-to-end latency in traffic scrubbing center

Poseidon can mitigate DDoS attacks effectively

Policy Expressiveness

```
syn_count = count(pkt.tcp.flag == SYN, [ip.src], 5)
ack_count = count(pkt.tcp.flag == ACK, [ip.src], 5)

if syr
    dr
else i
    pa
else:
    sp
```

```
dns_query = count(pkt.tcp.dport == 53, [ip.src], 3600)

if pkt.tcp.sport == 53:
    i

    e
```

```
http_get_counter = count(pkt.http == GET, [ip.src], 5)

if http_get_counter([pkt.ip.src]) >= T:
    puzzle
```

```
packet_byte = aggr(True, [ip.src], 5)
connection_number = count(pkt.tcp.flag == SYN, [ip.src], 5)
```

defense

```
udp_counter = count(pkt.ip.protocol == UDP, [ip.src], 5)

if udp_counter([pkt.ip.src]) >= T:
    rlimit
```

defense

```
packet_byte_counter = aggr(True, [ip.src, ip.dst,
          ip.protocol, tcp.sport, tcp.dport], 5)

if packet_byte_count([ip.src, ip.dst, ip.protocol,
          tcp.sport, tcp.dport]) >= T:
     rlimit
else:
     pass
```

Elephant flow defense

Policy Expressiveness

- Lines of Code

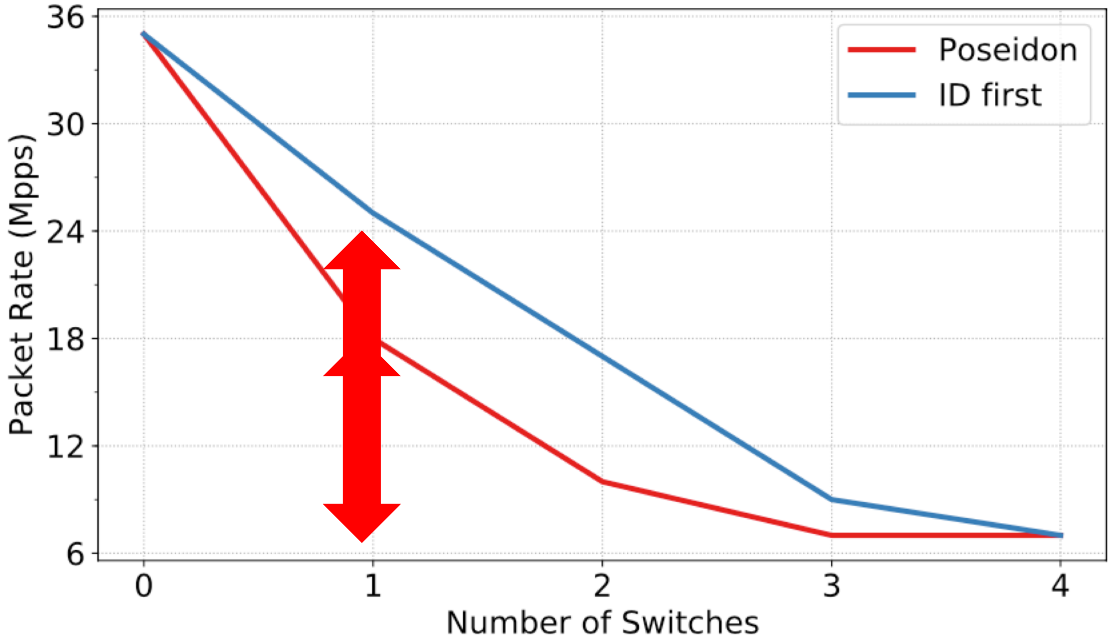
Policy	Attack	POSEIDON	P4	DPDK
1	SYN flood	9	939	1070
2	DNS amplification	7	255	898
3	HTTP flood	6	354	1184
4	Slowloris	8	513	995
5	UDP flood	6	376	911
6	Elephant flow	6	373	903

Policy Expressiveness

Protocol	DDoS attack	Description	Typical defense solution	Poseidon defense
ICMP	ICMP Flood	The victim servers are flooded with fabricated ICMP echo-request packets from a wide range of IP addresses	Rate-limit received ICMP packets from the same address or subnet	count + rlimit/pass
	Smurf Attack	A large number of fake ICMP echo-request packets with the victim severs' IP address are broadcast to a large network using an IP broadcast address	Filter ICMP echo-reply packets that are not queried by the victim servers	count + drop/pass
TCP	SYN Flood	The victim servers are bombarded with fabricated SYN requests containing fake source IP addresses	SYN Cookie/Proxy	count + sproxy/pass/drop
	SYN-ACK Flood	The victim servers are flooded with a large number of fake SYN-ACK packets	Filter SYN-ACK packets that are not queried by the victim servers	count + pass/drop
	ACK Flood	The victim servers are flooded with fabricated ACK packets from a wide range of IP addresses	Filter ACK packets that have not been responded by the victim servers with SYN-ACK packets	count + pass/drop
	FIN/RST Flood	The victim servers are bombarded with fake RST or FIN packets that do not belong to any of active connections	Filter FIN/ACK packets that do not belong to any action connections, then rate-limit received FIN/RST packets from the same connection	count + rlimit/pass/drop
	Attack			
	SSDP DDoS Attack	The attacker spoofs discovery packets with the victim servers' IP address to each plug-and-play device, to request for as much data as possible by setting certain flags	Filter SSDP replies that are not queried by the victim servers	count + pass/drop
	QUIC Reflection Attack	By spoofing the victims' IP address and sending a "hello" message to QUIC servers, the attacker tricks the servers into sending large amounts of unwanted data to the victim servers	Filter QUIC replies that are not queried by the victim servers	count + pass/drop
	NTP Amplification Attack	The attacker sends numerous NTP requests providing the victim servers' IP address	Filter NTP replies that are not queried by the victim servers	count + pass/drop
	Memcached DDoS Attack	The attacker spoofs requests to a vulnerable UDP memcached server, which then floods a targeted victims with large amount of traffic	Filter Memcached replies that are not queried by the victim servers	count + pass/drop
HTTP	HTTP Flood	The attacker generates large numbers of HTTP requests and sends them to the victim servers	Set limits for client sessions, CAPTCHA	count + pass/puzzle
	SlowLoris Attack	The victim servers are bombarded with too many open connections	Rate limit IP sources that establish numerous connections but send a few bytes	count/aggr + rlimit/pass

Poseidon can support a wide range of state-of-the-art DDoS defense mechanisms easily

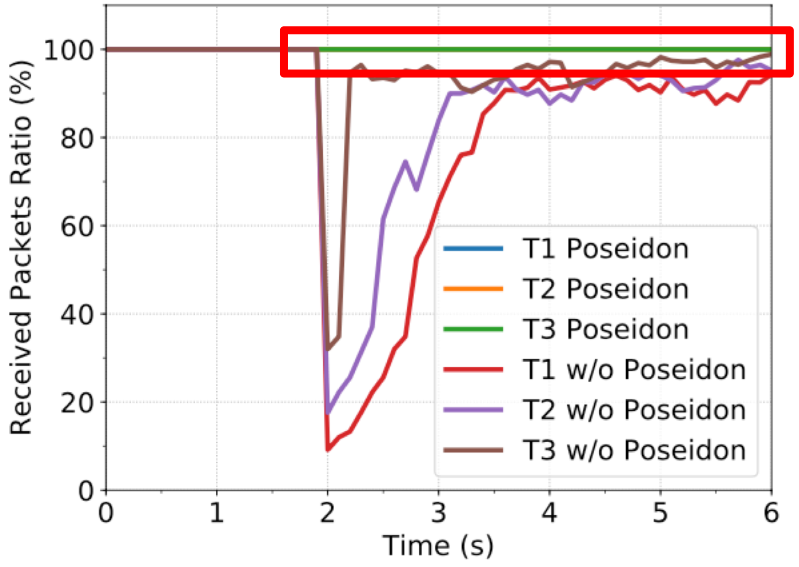
Policy Placement Mechanism



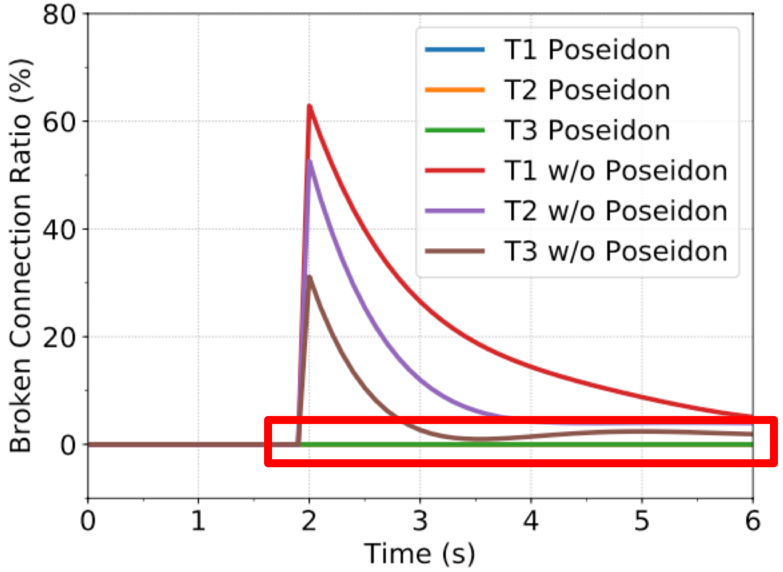
Traffic arriving at servers

Poseidon can orchestrate the defense resources efficiently

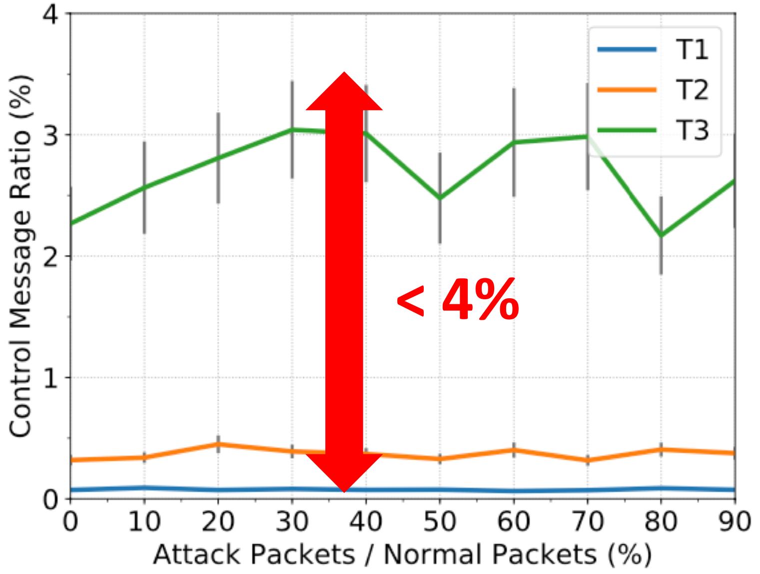
Dynamic DDoS Attacks



Received packets before/after policy transition (packet loss)



Broken connections before/after policy transition (flow interruption)



Control traffic/workload traffic ratio

Poseidon can cope with dynamic DDoS attacks effectively with minor overheads

Conclusion

- DDoS defense today: *expensive*, *inflexible*, and *low performance*
- Poseidon: programmable switches for *cost-efficient*, *flexible* and *performant* DDoS defense
- Key challenges: heterogeneity, resource constraint, dynamic
- Main solutions:
  - Simple, modular policy representation
  - Optimized, efficient defense orchestration
  - Handling dynamic attacks at runtime
- Highly effective in mitigating modern DDoS attacks

Thanks!
Q&A

zhangmh16@mails.tsinghua.edu.cn