Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches

Zaoxing Liu, *Boston University;* Hun Namkung, *Carnegie Mellon University;* Georgios Nikolaidis, Jeongkeun Lee, and Changhoon Kim, *Intel, Barefoot Switch Division;* Xin Jin, *Peking University;* Vladimir Braverman, *Johns Hopkins University;* Minlan Yu, *Harvard University;* Vyas Sekar, *Carnegie Mellon University*

https://www.usenix.org/conference/usenixsecurity21/presentation/liu-zaoxing

This paper is included in the Proceedings of the 30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the 30th USENIX Security Symposium is sponsored by USENIX.

Jaqen: A High-Performance Switch-Native Approach for Detecting and Mitigating Volumetric DDoS Attacks with Programmable Switches

Zaoxing Liu[⋆] Hun Namkung[§] Georgios Nikolaidis[†] Jeongkeun Lee[†]
Changhoon Kim[†] Xin Jin[◁] Vladimir Braverman[‡] Minlan Yu[◇] Vyas Sekar[§]
[⋆]*Boston University* [†]*Intel, Barefoot Switch Division* [◁]*Peking University*
[‡]*Johns Hopkins University* [◇]*Harvard University* [§]*Carnegie Mellon University*

Abstract

The emergence of programmable switches offers a new opportunity to revisit ISP-scale defenses for volumetric DDoS attacks. In theory, these can offer better cost vs. performance vs. flexibility trade-offs relative to proprietary hardware and virtual appliances. However, the ISP setting creates unique challenges in this regard—we need to run a broad spectrum of detection and mitigation functions natively on the programmable switch hardware and respond to dynamic adaptive attacks at scale. Thus, prior efforts in using programmable switches that assume out-of-band detection and/or use switches merely as accelerators for specific tasks are no longer sufficient, and as such, this potential remains unrealized. To tackle these challenges, we design and implement Jaqen, a *switch-native* approach for volumetric DDoS defense that can run detection and mitigation functions entirely inline on switches, without relying on additional data plane hardware. We design switch-optimized, resource-efficient detection and mitigation building blocks. We design a flexible API to construct a wide spectrum of best-practice (and future) defense strategies that efficiently use switch capabilities. We build a network-wide resource manager that quickly adapts to the attack posture changes. Our experiments show that Jaqen is orders of magnitude more performant than existing systems: Jaqen can handle large-scale hybrid and dynamic attacks within seconds, and mitigate them effectively at high line-rates (380 Gbps).

1 Introduction

Distributed Denial of Service (DDoS) attacks continue to be a destructive force in today's Internet [1]. Despite decades of work, *volumetric* attacks continue to be a severe threat, with growing attack volumes and types. In this respect, Internet Service Providers (ISPs), as the infrastructure to route Internet traffic, are at a unique vantage point to combat such volumetric attacks without interrupting client-side services.

In this context, programmable switching hardware has emerged as a promising means to enable defenses against volumetric DDoS attacks [2–7]. In particular, they promise better cost, performance, and flexibility tradeoffs, compared to traditional solutions. For instance, proprietary/fixed hardware appliances are expensive, have limited capabilities, and hard to upgrade in the field (e.g., [8, 9]). On the other hand, software appliances (e.g., [10]), while dynamic and reprogrammable, incur large latency, and are not efficient for large attacks. In addition, both classes of approaches entail high capital costs [9–11]. In contrast, programmable switches promise high line-speed guarantees (e.g., 6.5Tbps [12]), sufficient programmability (e.g., P4 [13]), and lower cost (Table 1).

Realizing this promise, however, is easier said than done, and the ISP setting creates unique and fundamental challenges that existing solutions do not address. Given that ISPs are *inline* and on the critical path of large attack traffic volumes, we need to support a *broad spectrum* of *detection* and *mitigation* natively on the programmable switches. Unfortunately, existing programmable switch-based solutions fail on one or more of these dimensions [3, 4, 6, 7, 14]. Specifically, existing efforts rely on out-of-band detection with the need to reroute traffic to separate monitoring infrastructure, which entails additional latency and cost [15–17]. Furthermore, many of these support a small number of mitigation functions [4, 6, 7, 14], or do so in an inefficient manner that exhausts the limited switch resources and can disrupt legitimate connections [3].

To this end, we present **Jaqen**, a *switch-native detection and mitigation system* that handles a broad spectrum of volumetric attacks [18] within ISPs. Unlike prior solutions, Jaqen completely runs on programmable switches (i.e., switch-native) and fully leverages their capabilities for accurate detection and fast response as attack postures change. Jaqen is an agile system that dynamically distributes detection and mitigation capabilities in a network-wide setting when available switch resources, attack types, and traffic volumes change.

Our overarching goal is to design a secure-yet-practical defense system, working within the limited switch chip resources (e.g., O(10MB) SRAM and limited accesses to the SRAM [12]). To see why this is challenging, consider two natural strawman solutions. First, to cover many attacks, we can consider running all potential detection and mitigation mechanisms on the switch. Unfortunately, this is infeasible due to resource constraints. Alternatively, we can run only a subset of detection and mitigation modules. However, this creates blind spots, where we do not have visibility into ongoing attacks, especially when attacks can dynamically change; i.e., the detection module checks for SYN floods but the attacker changes to a DNS amplification that goes undetected.

As a practical and robust alternative to these strawman solutions, we argue for a *broad-spectrum always-on detection* and *on-demand mitigation* design approach. That is, the detection logic must continuously (i.e., always-on) identify all attacks in our scope to avoid blind spots in face of dynamic attacks. Rather than enable all mitigation modules, we install them as needed (i.e., on demand) to optimize hardware resource usage. Given this high-level design philosophy, we address key algorithmic and system design challenges in Jaqen.

(1) ***Designing switch-native detection with high coverage:*** We build a switch-native, broad-coverage detector for ISPs by bridging universal sketch techniques in network monitoring [19, 20] and general DDoS detection. Instead of crafting multiple custom algorithms to achieve coverage (e.g., [15, 21–27]), universal sketches make it possible to track a broad range of current and unforeseen metrics with a single algorithm. We design the detector with two layers: *Data plane*—universal sketches as data plane primitives that can be pulled by the controller or configured as event triggers. *Control plane*—detection API for users to configure the sketches, query relevant metrics, and compute detection decisions.

(2) ***Flexible and performant switch-native mitigation:*** We identify a unified abstraction to implement mitigation with three interactive components: (1) *filtering* to drop, allow, or rate limit packets, (2) *analysis* to identify malicious traffic, and (3) *update* to the filtering when needed. For each component, we design a library of relevant mitigation functions with API based on best-practice mechanisms (e.g., intentional SYN drop [28] and DNS matching [23]) using switch-optimized logic and probabilistic structures [29–33]. Thus, constructing sophisticated (and possibly new) mitigation strategies will be like flexibly combining different building blocks on hardware using our API.

(3) ***Network-wide management to handle dynamic attacks:*** When attack postures change, Jaqen needs to compute a new resource allocation to redirect traffic to other available switches with the smallest rerouting cost. We formulate this as a Mixed-Integer Program (MIP). However, for large ISPs, a state-of-the-art solver could take a long time (10s of min) to finish. Thus, we design a responsive near-optimal heuristic that is 3-4 orders of magnitude faster.

We implement Jaqen in Barefoot Tofino switches [12] using the P4 language [13]. Our evaluation, performed on a set of one 6.5 Tbps programmable switch and eleven 40 Gbps servers, shows that Jaqen (1) accurately detects the attack type and estimates the attack volume with 97% accuracy when the attack traffic is not negligible (>1.5% of tested 380-Gbps throughput), (2) reacts to hybrid and dynamic DDoS attacks within 15-sec (including 10-sec detection period), and (3) mitigates the attack traffic with low false positives and negatives (varying from 0.0 to 0.072). Although our testbed only generates 380 Gbps traffic due to limited equipment, Jaqen with one switch can potentially handle Tbps-level attacks without interrupting legitimate users.

Contributions and roadmap. In summary, this paper makes the following contributions:

- Highlighting the requirements for ISP-based defense and identifying security limitations of existing P4-based defense solutions in the ISP setting. (§2)
- An integrated DDoS detection and mitigation framework entirely on programmable switches for defending volumetric attacks in ISPs. (§3)
- A broad-spectrum switch-native detector using universal sketching techniques and a library of highly optimized mitigation primitives for developers to write state-of-the-art and possibly new mitigation strategies in P4. (§4,§5)
- A network-wide resource manager that optimally deploys detection and mitigation modules in the network. (§6).
- An end-to-end system realization of Jaqen (§7) and demonstration of its effectiveness in handling real-world large-scale dynamic attack. (§8)

2 Background and Motivation

In this section, we begin by highlighting the requirements for ISP-centric defense and the opportunities that programmable switches bring. We then discuss existing defense solutions and highlight their shortcomings.

2.1 Requirements for ISP-centric Defense

ISPs own a large hierarchy of switches and routers to route user traffic to and from destinations, but usually do not access application-level user information. Given this nature, defending volumetric attacks in ISPs is appealing and ISP-based defense systems shall consider the following requirements:

- *Impact on benign traffic*: For service providers, the overarching goal is to improve user experiences for legitimate users. Thus, ISP-based defenses must not *interrupt* or *drop* legitimate user connections and shall not add large *extra latency* to benign traffic. Ideally, ISPs should limit the amount of traffic rerouted to out-of-band detection and scrubbing centers, and limit the usage of slow packet processing elements (e.g., servers) on the critical network paths.
- *Defense performance*: As a defense system, we need to support high packet processing capabilities to handle a broad range of existing and future attacks.
- *Cost efficiency*: As ISPs need to handle massive amounts of traffic every day (e.g., 100PB per day at AT&T in 2016 [37], we want to reduce the capital cost of defense devices and potentially their operational cost.

Opportunities of Programmable Switches. As observed in concurrent efforts [3–5, 7], modern programmable switches are appealing to augment DDoS defense performance. We envision these switches are promising in fulfilling require-

DDoS Solutions	Detection	Mitigation	Design	Performance (per unit)	Cost/Power
Bohatei [10]	No	Server-based	Full flexibility	10Gbps (80ms)	$5,600/600W
Arbor APS [34]	No	Cloud-based	Full flexibility	20Gbps (80ms)	$47,746/400W
ADS-8000 [35]	No	Hardware	Limited, hard to upgrade	40Gbps (<10ms)	$102,550/450W
FPGA-based [36]	Feasible	Hardware	Flexible, hard to program	4×25Gbps (<10ms)	$7,530/215W
Poseidon [3]	No[1]	Switch+Servers	Standard modules based on servers	3.3Tbps (12μs-80ms)	>$10,500/350W
Jaqen	In-band	Switch (ISP)	Switch-optimized logic/structures	3.3/6.5Tbps (12μs)	$10,500/350W

Table 1: Comparison of DDoS defense solutions. Top three are traditional solutions and the bottom two use programmable switches.

ments for ISP-scale defense: (1) *High line-rate guarantee* such as 6.5Tbps for any programs that fit in their hardware resources, which is appealing for combating large-scale attacks; (2) *Flexibility to support evolving attacks* while traditional hardware appliances are either fixed-function or have low programmability. With new switch architectures, we have the flexibility for both detection (e.g., capture packet signatures with the programmable parser) and mitigation (e.g., filter attack traffic with customizable rule tables; (3) *Cost-efficient* with cost similar to legacy switches of the same speed while having significantly lower capital costs than other appliances (e.g., a 6.5 Tbps switch costs around $12,000 [38, 39] while Arbor TMS [11]/APS [8] and Cisco Guard [9] cost $128,000 to $220,000 based on public estimates from [10]).

We provide a brief overview of programmable switch architectures for completeness. As shown in Figure 1, a representative programmable switch architecture is Protocol Independent Switch Architecture (PISA) [40], where the ASIC chip consists of a programmable parser and a number of reconfigurable match-action tables. Developers can program the packet parser to support user-defined packet headers, specify the matching fields and types (e.g., exact, range, and ternary matching), and configure supported actions (e.g., CRC hash, header field modification, register read/write via arithmetic logic unit (ALU), arithmetic operations using , and metering). We refer readers to §7 for more details.

2.2 Existing DDoS Defenses and Limitations

Table 1 highlights the tradeoffs between cost, performance, and flexibility in today's DDoS defenses.

Traditional DDoS defense solutions. At a high-level, traditional defense solutions include: (1) *Proprietary hardware* can be employed to differentiate suspicious traffic from legitimate traffic and filter out the attack traffic. However, there are key drawbacks. First, we need expensive appliances to deal with large-scale attacks. Second, they have low flexibility as they are hard to program and upgrade. (2) *SDN/NFV-based defense systems* have been proposed to detect and respond to DDoS attacks [10, 41, 42] that orchestrate available resources to dynamically allocate mitigation power for attacks. However, using software-only solutions is not scalable. Even if the operator has enough servers, benign traffic needs to be rerouted through a number of mitigation VMs, increasing the rerouting and processing latency. Moreover, the server footprint can be high; e.g., for a 100 Gbps DDoS attack, Bohatei may need 1000+ well provisioned VMs [10], which is not economical.

[1] Poseidon assumes given detection results. Poseidon has a monitor primitive, but its goal is to provide count/aggregation for known attack mitigation.

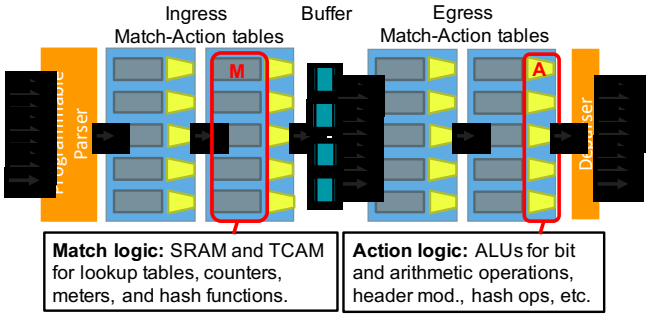


Figure 1: Protocol Independent Switch Architecture.

Programmable switch-based defenses. We consider a threat model (§3.1) where an adversary can launch dynamic attacks drawn from a set of popular volumetric DDoS attacks. Thus, ideally we need a defense system that achieves coverage over a broad spectrum of attack types and rapid response as attack situations change. Unfortunately, most existing efforts in switch-based defenses are based on the P4 behavior software simulator [43] with unrestricted resources and operations, except for recent efforts [3, 14] that have hardware implementations. These efforts suffer from one or more key limitations in ISP settings because of the non ISP-centric design:

- **Out-of-band and low-accuracy attack detection:** Most of these solutions, including Poseidon [3], essentially "punt" on the detection problem similar to the assumption in Bohatei [10]. Essentially dedicated NetFlow-like monitoring infrastructures (e.g., running on legacy routers and computing statistics with servers offline) are required to coalesce packet-level data into flow-level records. This may potentially offset the hardware cost savings that programmable switches could offer. While this was a reasonable assumption for an NFV-oriented deployment like Bohatei which envisions augments an existing network infrastructure, this is a somewhat ironic assumption for switch-based defenses. Even if we implemented these algorithms natively on the switch, they still incur limitations as (1) packet sampling approaches cannot provide fine-grained detection results [15, 44, 45] and (2) it requires

extra computation resources to conduct offline analysis, inducing significant detection delay.

- **Low-performance, in-effective mitigation:** Most existing efforts [4, 5, 7] build mitigation mechanisms covering only specific attack types such as SYN flood. While Poseidon [3] arguably has coverage on dynamic attacks, it does so by running backup defense modules on servers and reroutes *all* traffic to servers for state migration when attacks change, which is incompatible with the ISP scenario. More importantly, Poseidon's contribution is in designing switch-based mitigation for traffic scrubbing centers, where there might be a limited number of legitimate flows involved. When considering the ISP scenario with many legitimate flows, using Poseidon's switch component may not be as performant. For example, Poseidon uses a standard SYN proxy on the switch in a similar way as CPU by recording 65k legitimate sessions in a hash table. This default table will not scale to even hundreds of thousands of connections given the O(MB) on-chip memory constraint, and the hash collisions will cause the drop of many legitimate connections, just as a denial of service. Our experiments in §8.1 report 25% collisions for maintaining 2M legitimate connections (table size 2^{21}).

In summary, we see that while concurrent work has also argued the promise of programmable switches, ISP-based defense remains an open challenge: it is difficult to achieve the performance, flexibility, and cost benefits at the ISP scale.

3 Jaqen Overview

In this section, we describe the scope and architecture of Jaqen before we discuss the main technical challenges.

3.1 Problem Scope

Threat model. Our focus is on volumetric DDoS threats that aim to exhaust the available bandwidth and resources of the victims [1], such as TCP SYN flood, ICMP flood, Elephant flows, DNS flood, and other amplification threats including DNS, NTP, and Memcached. Other attacks such as nonvolumetric application-layer attacks or link flooding attacks [46] are outside the scope of this paper. We consider a *hybrid* and *dynamic* DDoS threat [1] that adversary can dynamically choose from a set of candidate attacks $\{A_i\}_{\forall i}$ at different times to launch a DDoS attack. The adversary has a volume budget V specifying the maximum rate that can be used to launch the attack at a given time. That is, $\sum_i v_t(A_i) \leq V$, where $v_t(A_i)$ is the volume of attack A_i at time t. Given such a budget V, the adversary can control the choice of type and volumes from set $\{A_i\}_{\forall i}$ to generate an attack.[2] We assume that programmable switches cannot be compromised by the adversary.

ISP deployment. We envision ISPs being early adopters of such a framework, given they are already adopting programmable hardware [47–49]. For instance, ISPs can deploy Jaqen in their network infrastructure to offer defense as a service to their customers. Our system can also coexist with other defense solutions (e.g., NFV, dedicated ASICs) at other locations to augment their capabilities against volumetric attacks; however, exploring this hybrid design is outside the scope of this paper.

[2] For instance, v_1(SYN)=50%·V and v_1(DNS)=50%·V at time 1, and then v_2(SYN)=10%·V and v_2(ICMP)=90%·V at time 2.

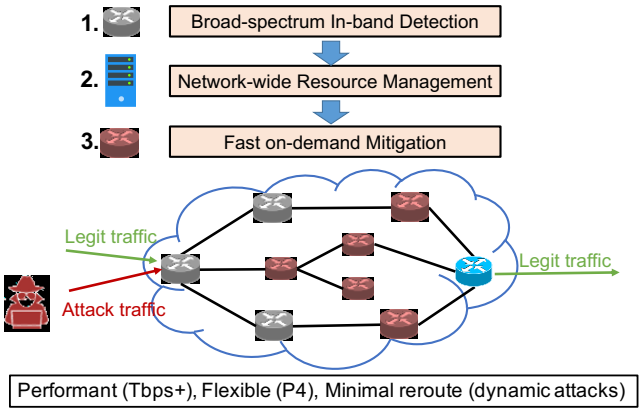


Figure 2: Overview of Jaqen

3.2 Jaqen Workflow

Jaqen has three logical steps, as presented in Figure 2:

(1) *Detection*: We do not assume prior knowledge if there is an ongoing DDoS attack. In this step, Jaqen provides information about whether protected users are under attack, what types and volumes of the attack are. During this step, the switch data plane identifies the suspicious traffic towards detected victims and report the estimated volumes of each attack type. An example output of this step is "victim=10.0.0.1, srcprefix=11.0.1.*+12.0.3.*, total= 2.5Gbps, vol=DNS(0.4)+SYN(0.3)+NTP(0.1)".

(2) *Resource management*: Once the detection information about the attacks is available, the resource manager on the controller makes resource allocation decisions on where to deploy mitigation based on attack detection results using minimized hardware resources.

(3) *Mitigation*: Based on resource management, the controller deploys mitigation modules onto the switches in the network. These modules effectively and accurately block attack traffic at packet arrival rates. After scrubbing the malicious traffic, the switches forward legitimate traffic without additional processing and network latency.

Attack coverage. Jaqen's primary focus is to enable defenses against a broad spectrum of volumetric attacks. Our definition of a volumetric attack is that the attacker sends a high amount of traffic or request packets to exhaust the bandwidth or resources of the victim. Our current Jaqen prototype handles 16 common volumetric attacks as described in Table 16:

- *TCP-based attacks*: SYN flood, ACK flood, RST/FIN flood, DNS flood (over TCP), TCP elephant flows, etc.

- *UDP-based attacks*: Amplification attacks using various UDP-based protocols—DNS, NTP, SNMP, SSDP, Memcached, QUIC, and UDP flood.
- *ICMP-based attacks*: ICMP flood, Smurf attack, etc.
- *Application-layer attacks*: simple unencrypted HTTP Get/-Post flood, SIP Register flood, etc.

Interestingly, we can further extend the coverage to some non-volumetric attacks by using Jaqen API described in §4, such as Slowloris, HTTP slow post, ARP cache poisoning, and DNS spoofing. We describe these extensions in Table 16.

Potential limitations. We analyze the potential system and security limitations of Jaqen. First, existing programmable switches used in Jaqen do not implement full packet parsing. Thus, any attack detection and mitigation requiring payload information cannot be supported. Second, Jaqen needs a few seconds to react to the attacks. An advanced attacker who smartly and frequently changes the attack types (e.g., <5s) can evade the defense. However, this potential evasion would require more computation/bandwidth and make it more difficult for attackers to conceal their identities (e.g., due to frequent traffic pattern changes), leading to alternative defenses such as IP filtering near the attack source.

3.3 Challenges

Given this workflow, we highlight the key design challenges that we need to address in the following sections.

Challenge I: Broad detection coverage on current and future volumetric attacks (§4). Programmable switches are constrained in terms of expressiveness compared to general-purpose servers [50] and also have limited resources. As an example, Barefoot Tofino switch [12] has $O(10)$MB SRAM, $O(1)$ ALUs, and $O(10)$ hash units.[3] Such resource constraints limit the possibility of fitting a large set of (complex) algorithms into switch hardware. Thus, a natural question is, how do we achieve broad-spectrum detection for many attacks?

Challenge II: Switch-optimized, resource-efficient mitigation (§5). Programmable switch's high performance guarantee comes with constrained hardware resources and computational model. Best practice mitigation mechanisms designed for servers do not work well for programmable switches (e.g., not scalable and dropping legitimate connections) and we need to carefully craft mitigation functions to deliver envisioned high-performance protection to the users.

Challenge III: Efficient ISP-scale defense for dynamic attacks (§6). In an ISP, attack traffic can enter the network from arbitrary ingresses. One alternative is to deploy Jaqen modules only at the ingress switches on the edge. However, given the limited resources at switches (and other concurrent services on the switches), this may not be feasible. To this end, we propose to leverage other switches that have available resources to offer an ISP-scale network-wide defense while minimizing the total resource usage. When attack posture changes, we need to quickly react by recomputing a resource allocation that has minimal changes from the previous allocation. However, this means that we need fast resource allocation decisions, especially in large-scale networks, with minimal disruptions to ongoing traffic.

[3] The actual numbers are proprietary under switch vendor's NDA.

Detection Metric	Description	Poseidon	Jaqen
Count/Aggr. [30]	Count/Aggr. over a flow	✓	✓
Entropy [51]	Identify anomalies/attacks	×	✓
Distinct flows [52]	Distinct TCP/UDP flows	×	✓
Traffic change [53]	Heavily changed flows	×	✓
Signatures	Volumes of special packets	×	✓
New metrics	Arbitrary G-sum in [54]	×	✓

Table 2: Poseidon vs. Jaqen in supported detection metrics.

4 Efficient and General Detection

Programmable switch resources are constrained compared to x86 servers, which impose restrictions on supporting a broad spectrum of algorithms as x86. Thus, for ISP-scale detection running completely in switches, we want our detection module to be as compact as possible while having good coverage of attacks. Achieving both requirements is challenging as fitting many detection algorithms (e.g., [15, 21–27]) for coverage in the switch is infeasible. Instead, we observe that recent advances in universal sketching [19, 20, 55] for network monitoring can play a crucial role in designing a general DDoS detector. Conceptually, universal sketches are a class of approximation algorithms that can simultaneously estimate a range of network statistics supported by custom algorithms, e.g., heavy hitters [20, 27, 53, 56, 57], distinct flows [20, 58, 59], and entropy [60–62]. More precisely, a universal sketch is able to estimate any aggregated functions from a data stream that are asymptotically bounded by the L2 norm of the data [19], while recent switch-based approaches only support counting/aggregating flow sizes based on Count-Min sketch [30]. We summarize the major differences between Poseidon's monitor [3] and Jaqen's detector in Table 2.

To bring universal sketching into ISP-centric detection, we design an approach that has data plane and control plane components:

Switch layer: "Future-proof" universal sketches. As shown in Figure 3, the switch layer contains multiple universal sketches, complemented by a signature-based detector. Together, Jaqen has the ability to estimate a variety of current and possibly unforeseen metrics that are relevant to the attacks (i.e., future-proof), laying the foundation for accurate attack detection. For instance, the entropy value changes in terms of the srcIP and dstIP are a strong indicator of an ongoing attack; the difference between the numbers of DNS requests and responses hints a DNS-related attack. It is worth noting that some attack-related metrics that require cryptography data (e.g., Malformed SSL Flood) or require complete payload parsing (e.g., Zorro attack [63]) are outside our scope due to

Mitigation API	Description	Switch Design
`RateLimit(identity,rate)`	Rate limiter for flows that match certain rules	SRAM + meters
`ExactBlockList(identity,size)`	Blocklist to drop packets that match certain rules	SRAM + TCAM
`ExactAllowList(identity,size)`	Allowlist to allow packets that match certain rules	SRAM + TCAM
`ApproxBlockList(identity,config)`	Approximate block list to drop packets	LRU lossy hash table
`ApproxAllowList(identity,config)`	Approximate allow list to pass packets	Blocked Bloom filters
`ActionAndTest(action,List(predicate))`	Perform a packet action and test later w/ predicates	BF + action/control
`HeaderHashAndTest(identity,action)`	Perform header field hash and test w/ action	Cookie + action/control
`UnmatchAndAction(action,List(predicate))`	Find unmatched predicates and perform an action	CBF + action/control
`KVStore(key,value,size)`	A high-performance small key-value store	Hash-based KV store
`ReportCtr(identity,type)`	Update a filtering list via controller	Mirror to CPU
`Recirculate(identity,type)`	Update a filtering list via packet recirculation	Mirror and recirculate

Table 3: Jaqen's Mitigation API.

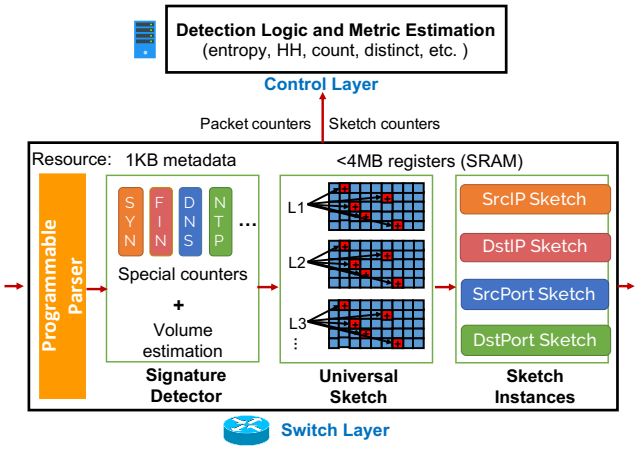


Figure 3: Switch detection design w/ universal sketches.

hardware limitations and the ISP-centric view.

While a canonical implementation of universal sketch is already more resource-efficient than a combination of multiple custom algorithms [20], we further reduce the resource footprint when there are multiple instances of universal sketches. In particular, our implementation follows the same trajectory of parallel efforts in optimizing universal sketching for other hardware domains [27, 64–66]. The full discussion is outside the scope of this paper; but at a high-level, we include the following three optimizations: (1) *Reducing hash computations* by consolidating short hashes into long hashes and reusing hashes across universal sketch instances [65]. (2) *Reducing memory accesses* by updating only one instance of Count Sketch (CS) [31] in universal sketch [64]. We reduce these hashes and memory accesses by updating only one CS per packet. (3) *Reducing flow key storage space* by using a two-way hash table as a cache, similarly as [66]. The flow keys are used to identify elephant flows or heavily changed flows.

By applying these optimizations, the resource usage of the universal sketching component has been significantly reduced by more than 50% as shown in the evaluation Table 5.

Control layer: Detection API and logic. Now we have the ability to obtain attack-related sketch counters from the switch

```
def Query ( proto, func, mode, freq ):
    try: // connect to switch
        received = conn_mgr.init ( )
        sleep ( freq )
        if mode == 0: // pulling mode
            p = read_registers ( proto )
            metric = offline ( p, func )
        elif received == 1:
            r = buffer_triggers ( proto )
            metric = offline( p, func )
        reset ( )
        return metric
    except Error:
        print ( "Switch access failed!" )
```

```
def UDPFlood (threshold):
    try:
        while 1: // query heavy hitters
            hhs = Query ( udp, src_hh, 0, 5 )
            for srcIP in hhs:
                if srcIP >= threshold:
                    report_to_ctr ( )

def DNSFlood (threshold):
    try:
        while 1: // unmatched DNS replies
            diff = Query ( udp, dns, 0, 5 )
            if diff >= threshold:
                report_to_ctr ( )
```

Figure 4: Simplified detection API and logic examples.

layer. These counters are reported to the control layer for offline metric estimation and running detection logic, as depicted in Figure 3. In the control layer, we need to figure out how to use these metrics for detection, e.g., what is the reporting mode (controller pulling or self reporting), what are the needed metrics, and how to realize a detection logic with supplied metrics.

To that end, we design an API with `Query(proto,func,mode,freq)` to precisely obtain the metrics for detection, where *proto* defines the queried protocol, *func* defines the function of the metric, *mode* defines the reporting mode (e.g., self-triggering with a threshold), and *freq* is the reporting frequency. For example, we can query the UDP srcIP heavy flows above a 0.5% threshold every 5 seconds in a self-reporting mode. After configuring the way to obtain metrics, we need to implement a detection logic to make detection decisions, such as detection for UDP flood and DNS flood (Figure 4).

5 Performant and Flexible Mitigation

We demonstrate the flexibility of Jaqen by providing a flexible P4-based mitigation API to construct switch-native mitigation strategies. In particular, in an ISP-scale, directly adopting standard server-based mitigation methods will not work. Instead, we need to convert a mitigation strategy into a switch-optimized one. To achieve this, we observe that mitigation strategies can be abstracted with three components that are interacting with each other, as shown in Figure 5. For each component, we design a set of mitigation functions that are

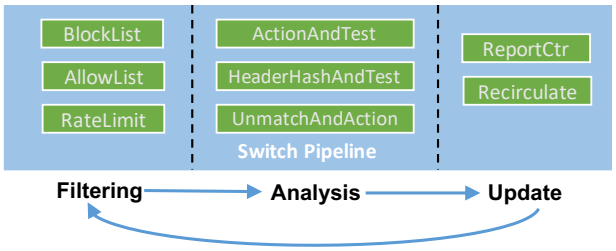


Figure 5: Abstraction of mitigation strategies.

optimized for switch resources based on state-of-the-art approaches. In total, we provide 11 building blocks to construct a broad range of mitigation strategies for the switches in ISPs.

1. Filtering: In a mitigation strategy, we first need to provide functions to block, rate limit, or allow packets that meet certain rules. For instance, a blocklist can drop packets from some malicious source IPs while an allowlist can directly pass the traffic from certain users (e.g., VIPs). In this component, we provide five functions as the following:

- `ExactBlockList/ExactAllowList(identity,size)` are two types of lists to drop or allow packets that exactly match a *flow identify* (e.g., srcIP, 5-tuple, or subnet). For example, blocking any traffic from srcIP 10.0.0.1. We encapsulate the exact match tables provided by the switch hardware to construct these two functions. Due to the switch memory constraint, the *size* of rules is usually limited to O(10K) per processing stage in switch pipelines. These exact lists are particularly useful when a small set of "VIPs" or "malicious clients" are known.
- `ApproxAllowList/ApproxBlockList(identity,config)` provide approximate allow- and blocklists. They offer the same functionality as `ExactAllowList/ExactBlockList(identity, size)` but can scale to O(10M) rules (depends on the *config*) if some approximation errors are acceptable. While errors are unavoidable, our design goal is not to let the errors affect legitimate ISP users. We achieve so by leveraging the features of the approximate data structures: (1) In the approximate allowlist, we use blocked bloom filters (w/ one hash function per block) to save switch resources. Bloom filers will only create *false positives* that may allow a small portion of attack traffic to pass through, while the legitimate connections are always allowed. (2) Similarly for the approximate blocklist, we design an LRU-alike lossy hash table in the switch, leading to only *false negatives* from the structure. The false negatives in a blocklist mean that some attack traffic might not be blocked while legitimate traffic (not on the list) remains unaffected.

  Hardware constraints: To implement the above structures, we need to store flow identities in *register arrays* using switch SRAM. The size of a register is upper bounded by a certain limit (e.g., 64-bit). To store flow identities that are larger than this limit (e.g., 5-tuple), we need to use multiple register arrays to store them, or replace the actual identities with hash values. While using hashed flow indices is a common practice, it may bring additional errors.
- `RateLimit(identity,rate)` maintains a rate limiter table with flow identities and user-defined rates. We use the built-in *meter* primitive in P4 to mark the flows with different colors and perform different rate controls based on the colors; e.g., green→no action, yellow→user rate, and red→drop.

2. Analysis: In the filtering step, some traffic has been marked as "allowed" or "blocked" and will bypass other functions in the switch for forwarding or dropping. For the unmarked traffic, we need to analyze whether the traffic is benign or not using designed four analysis functions.

- `ActionAndTest(action,List(predicate))` is a method to perform an *action* on a packet and analyze if succeeding packets match a list of pre-defined *predicates*. The supported actions here are switch embedded actions such as *drop* and *forward*. For instance, we use this API call to implement a best-practice mitigation function of intentional SYN drop (DropFirstSYN) [28]. This function is to filter out the malicious SYN traffic and prevent the switch being directly exploited as a reflector/amplifier using spoofed srcIPs. Specifically, for every SYN packet, the switch checks if it is a first-time SYN or a retransmitted SYN within 5 seconds (predicates). If the SYN is a first-timer, a drop action will be performed; the packet will be allowed to pass otherwise.
- `HeaderHashAndTest(identity,action)` defines a method to compute hashing on the *flow identify* (e.g., 5-tuple) of a packet and perform a test *action* with the hash. For example, the switch can produce a "cookie or nonce" by hashing the 5-tuple header fields and constructing a reply packet with the nonce. As a case study, we will use this primitive to design two types of switch-optimized SYN proxy/cookie mechanisms in the later "Case study" section.
- `UnmatchAndAction(List(predicate),action)` implements a function to test if a list of *predicates* are matched and then perform a packet *action* based on the matching result. Besides *drop* and *forward*, two additional actions are supported: *insert to*/*delete from* a probabilistic structure—counting bloom filter (CBF) [32]. In particular, we can use this function to realize an effective mechanism [23] to mitigate amplification attacks following specific protocols (e.g., DNS, NTP, SNMP). When some predicates are matched — protocol matches `<UDP,src=10.0.0.*,port=53>`, packet type matches `<QR=0>`, error field matches `<RCODE!=0>`, the packet identity will be inserted to the CBF as a valid DNS request. If a DNS reply matches `<UDP,dst=10.0.0.*,port=53>` and `<QR=1>` in the CBF, the packet identity will be deleted from the CBF. Any succeeding unmatched DNS replies (above a threshold) will perform action *drop*.
- `KVStore(key,value,size)` provides a small efficient key-value store using hash-based exact-match tables [67, 68].

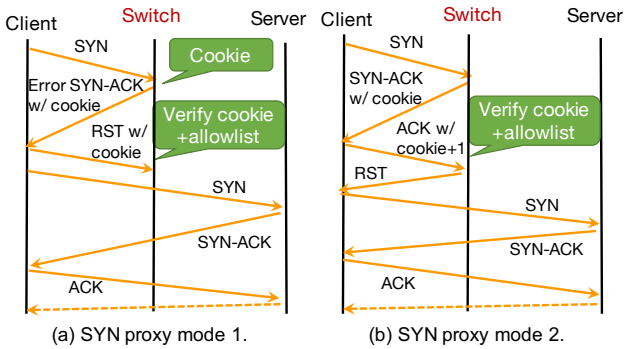


Figure 6: On-switch SYN Proxy workflows.

We can treat it as a high-performance, trustworthy registry service for certain protocols. For instance, we can leverage this function to build a high-performance DNS cache.

3. Update: After the analysis step, the (suspicious) traffic has been marked with a label (e.g., benign). As the final step of the mitigation, we may want to update an allow/blocklist or rate limiter to allow/block or rate limit the succeeding traffic from that flow. Since the filtering functions are placed ahead of the analysis components in the switch pipeline, we need either the switch controller or packet recirculation to update a list, as the following two API calls.

- `ReportCtr(identity,type)` requests to update one *type* of the filtering lists (i.e., blocklist, allowlist, and rate limiter) via switch controller. Specifically, the packet will be mirrored to the controller CPU via a dedicated PCIe lane and write information (*identity*) back to the switch data plane via the control API.
- `Recirculate(identity,type)` implements a similar update functionality without going through the switch controller. Specifically, this API function modifies a mirrored packet and recirculates it to the ingress port to update a filtering list with the required flow identity.

Hardware constraints: When using these two update functions, we as developers need to pay close attention to the hardware constraints: (1) The PCIe lane between the switch data plane and the control CPU has limited bandwidth (e.g., 100Gbps). It is impossible to process every packet on the controller, especially when the throughput is at a Tbps level. Thus, as shown in our mitigation examples later, we update the allow/block/rate-limit lists only when necessary. Take SYN flood mitigation as an example; we do not perform an update for every attack flow but update the allowlist only when legitimate clients pass DropFirstSYN and SYN Proxy tests successfully. (2) Packet recirculation affects the switch processing capability. For example, recirculating every packet will halve the total capability. We can perform recirculations without performance degradation when the effective throughput is lower than the switch limit. But as a general rule, any mitigation strategy should control the expected number of recirculated packets.

Case study: Design switch-native SYN proxy using the API. SYN Proxy/Cookie is a best-practice method to mitigate SYN flood attack using a server as a proxy for shielding malicious SYN traffic. The typical workflow of a SYN proxy can be described as: (1) When a SYN is received, the proxy server generates a unique `cookie` [69] with 5-tuple and adds it to the sequence number (`seq. no.`) header field of the corresponding SYN-ACK reply. (2) When a legitimate client receives the SYN-ACK it will acknowledge back an ACK packet with `cookie+1` in its seq. no.; Otherwise, an attacker would not send the cookie back. (3) Once the proxy verifies the correctness of the cookie, it will record session information (e.g., seq. no. difference) and construct a new SYN to the designated destination to establish the connection. The succeeding packets will go through the proxy to translate the seq. no. in order to continue the original TCP handshake.

Unfortunately, the current switch-based SYN proxy that directly implements the above server-based design (e.g., Poseidon [3]) has scalability issues when there is a large number of legitimate connections. They maintain seq. no. translation data for *each* legitimate connection using a single hash table (e.g., size 65536). Inevitably, using a single hash table for per-connection state storage would break the correctness of many legitimate connections due to hash collisions. For instance, keeping 65536 legitimate connections on a hash table of size 65536 has expected 24109 collisions.[4] To address this issue, we design two SYN proxy modes with `HeaderHashAndTest(identity,action)` to perform a "cookie" operation on designated header fields with hashing and send back a response packet (e.g., SYN-ACK) to "test" if the client is legitimate. Note that in our design, one can use `ApproxAllowList(identity,config)` to record a large number of legitimate identities that have passed the tests. The approximation errors will not affect legitimate traffic since Bloom filters do not create false negatives.

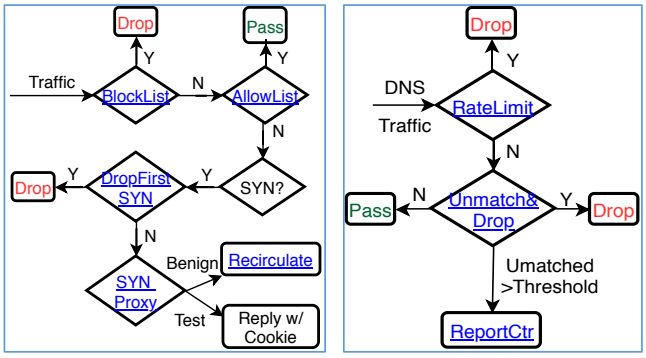
- `SYNProxyMode1` as depicted in Figure 6(a): (a) When the switch receives a SYN, it will generate a cookie to be added in the seq. no. header field while modifying the acknowledgment number field to a large out-of-window number (e.g., $+2^{18}$). (b) When the client receives such a SYN-ACK with a wrong ack. no., it realizes issues in the current TCP handshake and generates an RST with the received seq. no. (cookie), according to standard TCP specs [70]. (c) When the RST packet is received by the switch and the cookie is verified, the connection identity (e.g., 5-tuple) will be added to an allowlist. Then the client will retry to establish a connection. Note that this proxy can also be used as mitigation for DNS traffic carried over TCP.

  Extra connection setup time: This mode requires the client to retry a SYN to establish the connection. As we show in Table 4, most legitimate clients should retry the

[4] The expected collisions for a sequence of n values and a hash function of m values, can be calculated using birthday paradox as $n-m+m(\frac{m-1}{m})^n$.

Client	Retried Conn.	Setup Time
Wget (Ubuntu)	✓	2.1s (Local Testbed)
Curl (Ubuntu)	✓	2.1s (Local Testbed)
Chrome (Ubuntu)	✓	1.5s (Local Testbed)
Chrome (Android)	✓	1.8s (Campus VPN)
Safari (iOS)	✓	1.3s (Campus VPN)
Firefox (Windows)	✓	1.9s (Campus VPN)

Table 4: Connection setup time in SYN Proxy Mode 1.



(a) SYN flood mitigation (b) DNS amp. mitigation

Figure 7: Mitigation strategy examples.

connection. In fact, a retried SYN is a best-practice indicator to flag legitimate connections ([28]).

- `SYNProxyMode2` is an alternative SYN proxy design on switches. In `SYNProxyMode1`, there can be a wait time between the RST (w/ cookie) is sent from the client and the client starts to retry the connection with a new SYN. This wait time is usually not long (within a few seconds) depends on the client implementation. To reduce this wait, we also consider an alternative design that asks the client to resend the SYN immediately, as shown in Figure 6(b). This mode uses the same logic as original SYN proxy but will reply an RST to the client once the cookie is verified. In the midst of an unsuccessful connection initialization, the client usually resends a SYN to reestablish the connection once an RST is received.

Mitigation strategy examples. Using our mitigation API, Jaqen supports sophisticated and best practice mitigation strategies for volumetric attacks. We briefly summarize a broad range of volumetric attacks and Jaqen's mitigation in Table 16. To illustrate the use of the API, we describe two representative examples on mitigating SYN flood and DNS amplification with sophisticated defense strategies, as depicted in Figure 7. In addition to DNS amplification, Figure 7(b)'s workflow can be applied to other amplification attacks. Both examples fit in a single switch pipeline.

In SYN flood, the suspicious traffic will first go through a **`BlockList`** and an **`AllowList`**. If the packet passes the lists and is a first-time SYN, we drop it via **`ActionAndTest(drop, firstsyn)`** (DropFirstSYN) to reduce the traffic for SYN proxy analysis. If this SYN is not a first-timer, we analyze if it is

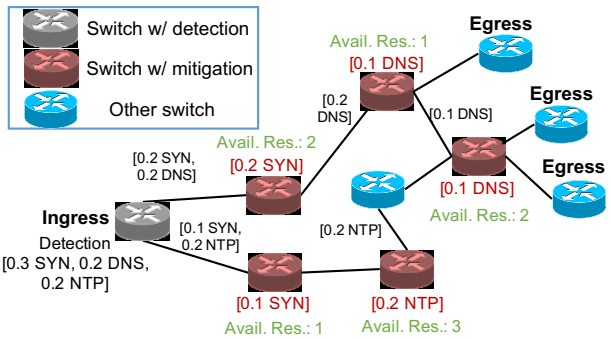


Figure 8: Example network-wide resource management on a simplified Claranet topology [71].

from a legitimate client via **`HeaderHashAndTest(conn,synack)`** (SYNProxy). If this client passes the SYN cookie analysis, we will update the allowlist accordingly using **`Recirculate`**. The succeeding traffic from this legitimate client will be allowed without going through the analysis modules.

In DNS amplification, DNS traffic will go through a rate limiter to control the per-source rates using **`RateLimit`**. If a DNS reply has not been requested from a valid client before, we mark this packet as "unmatched" and drop it via **`UnmatchAndAction(drop,dns)`**; Otherwise, we will forward this reply to the destination. Once the unmatched DNS replies from certain sources exceed a threshold (using detection), we will update the rate limiter to control the rate from these sources via **`ReportCtr`**.

6 Network-wide Resource Manager

In an ISP, Jaqen needs to deploy mitigation modules as they are needed in the network, with minimized possible switch hardware resources. This minimization is to help preserve resources for other ISP services and to reduce the number of deployed modules for faster reaction. We describe a resource allocation problem that Jaqen needs to solve as a mixed integer program (MIP) and present a heuristic algorithm that returns a near-optimal allocation. As an example, Figure 8 shows the mitigation resource allocation on a simple topology. Compared to the solver-based allocation, our algorithm achieves better responsiveness given the scale of an ISP.

Problem inputs. Our basic assumption in the network-wide setting is that the ISP has available hardware resources on the path for potential DDoS attacks; we cannot mitigate the attacks otherwise.

- *Network topology and mitigation modules:* We define the ISP network topology as an undirected graph $G = (V, E)$, where V is the set of all switch nodes that carry network traffic and have enabled programmability and E is the set of interconnected links. We define $V = \{v_1, v_2, \ldots, v_n\}$ where each v_i is a vector of available information about Switch i. For instance, $v_i =< AvailRes_i, Band_i, Type_i >$ where $AvailRes_i$ is the number of available programmable pipelines, $Band_i$ is the allowed bandwidth, and $Type_i$ is the

Minimize: $TotalRes$, subject to

$$TotalRes = \sum_i \sum_j Alloc_{i,j} \quad (1)$$

$$\forall i : \sum_j Alloc_{i,j} \leq AvailRes_i \quad (2)$$

$$TotalAvailRes = \sum_i AvailRes_i \quad (3)$$

$$\forall i, j : Alloc_{i,j} \in \{0, 1, \ldots, TotalAvailRes\} \quad (4)$$

$$\forall d, e, k : N_{d,e,k} \subseteq \{1 \ldots i\} \quad (5)$$

$$\forall d, e, k : \sum_i \sum_j Alloc_{i \in N_{d,e,k}, j} * MCap_{i,j,k} \geq AttackVol_{d,e,k} \quad (6)$$

$$\forall e, d, k : \sum_i \sum_j Alloc_{i \in N_{d,e,k}, j} * BCap_{i,j,k} \geq ResVol_{e,d,k} \quad (7)$$

$$\forall i : \sum_j \sum_k Alloc_{i,j} * (MCap_{i,j,k} + BCap_{i,j,k}) \leq Band_i \quad (8)$$

Figure 9: MIP to compute optimal resource allocation

hardware type of the programmable switch. Further, the mitigation capability of module j on switch i for attack k is given as $MCap_{i,j,k}$ and the processing capability for reverse traffic is $BCap_{i,j,k}$.

- *ISP routing and Traffic Engineering information:* We assume the ISP has a controller that maintains and implements the routing and traffic engineering decision for all the network traffic that passes through the ISP. For instance, in a software-defined network (SDN), the (virtually) centralized controller maintains the routing decisions for each network flow on each switch. In our network-wide setting, we have the aggregated traffic distribution information at the controller level, which is defined as $Band_{i,k}$ for switch i, traffic type k , e.g., $Band_{i,\text{DNS}} =< DNS = 0.3 >$ and $Band_{i,\text{SYN}} =< SYN = 0.2 >$.

Problem statement. Given the problem inputs from the ISP, we define our network-wide resource allocation problem. Intuitively, based on the existing traffic distribution, we want to minimize the usage of hardware resources while still cover all attack traffic from all ingresses. We define a MIP formula in Figure 9 with constraints and definitions described below:

- Eq. (1) defines the total allocated resource as $TotalRes$, which is the aggregation of the resource allocated for each $switch_i$ and $module_j$.
- Eq. (2) ensures that the module resource allocation on any switch will not go over the available resource budget.
- Eq. (3) defines the total available resource as the sum of the available resources on all switches.
- Eq. (4) defines the number of allocated $module_j$ as an integer from total available resource.
- Eq. (5) defines the switch set that route the traffic of Attack k from Ingress d to Egress e. This information is given from routing decisions and detection results on the controller.
- Eq. (6) captures all the allocated mitigation modules on the ingress-egress path (d, e) and ensures the attack traffic on the path has been taken care of.
- Similarity, eq. (7) captures all the allocated modules on the egress-ingress path to make sure the response traffic has been handled. Eq. (8) confirms the capacity of allocated modules does not exceed the processing bandwidth.

Algorithm 1 Greedy Algorithm for Resource Allocation

Inputs:
Topology graph $G = (V, E)$ with IN as the ingress set and EG as the egress set
Routing info $Route_i$ for each switch
Ingress $d \in IN$, egress $e \in EG$, and attacks $k \in K$
$\forall d, e, k$: $N_{d,e,k}$
$\forall i \in |V|$: $AvailRes_i$
$\forall d, e, k$: $AttackVol_{d,e,k}$ and initialize $AttackVol_{d,e,k,i}$
$\forall d, e, k$: $ResVol_{e,d,k}$ and initialize $ResVol_{e,d,k_i}$
$\forall k, i, j$: $MCap_{i,j,k}$ and $BCap_{i,j,k}$
procedure GREEDYHEURISTIC($D(m, n)$)
    for d in IN **do**
        for e in EG **do**
            BFS with $AttackVol_{d,e,k}$ and $Route_i$
            $\rightarrow$ update $AttackVol_{d,e,k,i}$
            BFS with $ResVol_{e,d,k}$ and $Route_i$
            $\rightarrow$ update $ResVol_{d,e,k,i}$
    Sort the (d, e) paths P by total volume of the attacks.
    for p in P **do**
        Sort $N_{d,e,k}$ by $AttackVol_{d,e,k,i \in N_{d,e,k}}$
    for d in IN **do**
        for e in EG **do**
            Update $Alloc_{i,j}$ with $MCap_{i,j,k}$ and $BCap_{i,j,k}$
    Output: $\forall i, j$: $Alloc_{i,j}$

Fast mitigation module allocation. We design a greedy heuristic to achieve real-time mitigation module allocation. We present the pseudocode of the heuristic in Algorithm 1. The high-level intuition is the following: For each pair of ingress and egress that has potential attack traffic, we use Breadth First Search (BFS) with the given routing decisions to find the attack volume distribution on each of the switches. We then sort the switches along the path by their hybrid attack volume and allocate the mitigation modules to cover the largest volumes first in a greedy manner.

Updating mitigation modules for dynamic attacks. When mitigation modules need to change due to dynamic attacks, Jaqen follows a three-step procedure to update a switch: (1) *Rerouting:* the controller disables the filtering components on all activated switches and then computes and distributes new forwarding rules with the current switch excluded, in order to reroute the legitimate traffic on this switch. (2) *Replication:* Once the new rules have been applied, replicate the switch states about the legitimate connections (if not expired) in the

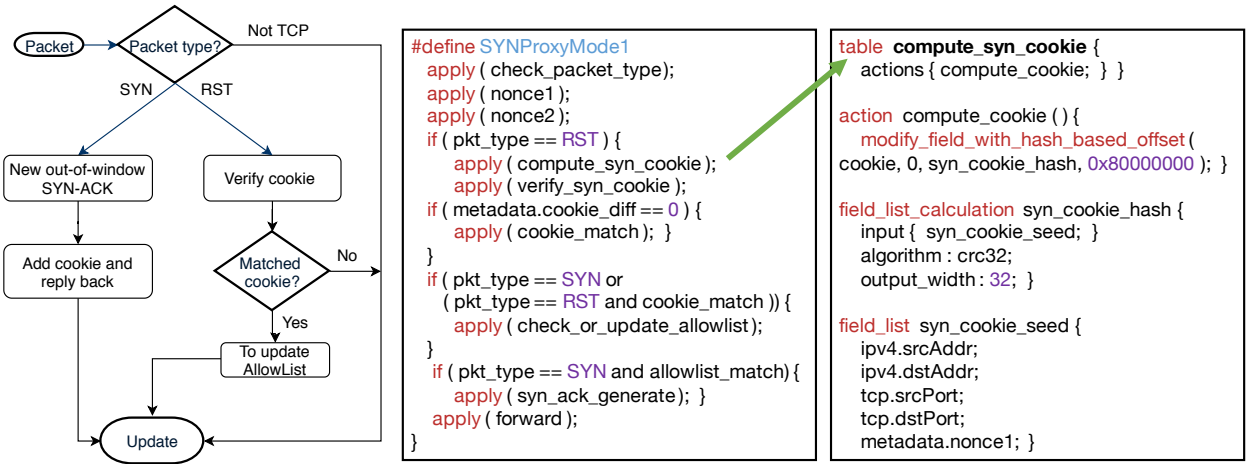


Figure 10: L to R: (1) SYN proxy mode 1 workflow (2) Abstract P4 code (3) SYN cookie example.

Impl.	Match Units	Hash Bits	SRAM	Action Slots
Original [20]	245	322	50	133
Our impl.	60	151	46	41

Table 5: Resource utilization of a universal sketch.

controller. (3) *Swapping:* reprogram the switch with the new set of modules and required states. Report to the controller to include this switch into the forwarding rules.

7 Jaqen Implementation

We have implemented a Jaqen prototype based on Barefoot Tofino using P4-14 for switch modules and using Python for switch controller. For P4 code compilation, we use Barefoot P4 Studio SDE [72]. In this section, we briefly describe how we implement the detection and mitigation API and demonstrate the convenience for developers to build new defenses. We open-source the prototype of Jaqen in [73].

Detection API and logic. To implement `Query(proto,func,mode,freq)`, we need multiple universal sketches [20] and signature-based counters in the switch data plane. We implement a universal sketch using a smaller number of ALUs than its original model implementation. As presented in Table 5, we achieve better resource efficiency by combining redundant sketch constructions and merge multiple hash computations and register operations into a single ALU operation. These optimizations are chosen depending on the specific resource numbers from Barefoot Tofino switch [12] and the accuracy guarantees we want to achieve. Thus these configurations are subject to change for other types of programmable switches.

When implementing signature-based counters, we write custom packet parsers to count some particular packets (e.g., TCP SYN, ICMP, and UDP DNS requests). Based on the configured *mode* and *freq*, the counters in the switch will either be self-reported or pulled by the switch controller. We will use these counters to compute attack-specific signatures (e.g., the number of unacked/terminated SYN requests) based on detection logic.

Mitigation API. We implement mitigation API using P4 and macro functions with several underlying structures described below. We also give an example workflow of Jaqen's SYN proxy and its abstract code in Figure 10. We refer reads to the project repository for more examples.

1. *Blocked Bloom filters:* We split Bloom filter's single array into multiple registers as a blocked Bloom filter. This split will maintain asymptotically the same error bounds [29]. We implement the blocked filter with one CRC32 hash per block of 1-bit registers. In each switch pipeline stage, we parallelize multiple blocks of filters for resource efficiency.
2. *Counting Bloom filters:* The goal of CBF is to record the inserted flow identities while supporting deletions from the filter. We implement CBF using an efficient two-part structure, where the controller maintains a complete CBF with 8-bit counters and the switch data plane stores an equivalent bloom filter with 1-bit registers.
3. *LRU cache:* We implement a lossy hash table with multi-layer of Least Recently Used (LRU) caches (r register arrays of d entries). When insertion, we hash the item to select one of d columns to conduct a rolling replacement of the r entries in the column by replacing the first one with the new entry, the second with the first, the third with the second, etc. When query, we check if the current item appears in one of the corresponding r entries in one of the d columns based on the hash.
4. *Key-value store:* We implement a key-value store based on the P4 logic of [68]. We offer a store that stores up to 64K entries with 16-byte keys (up to 64-byte) and 128-byte values. This store can be used for DNS or ARP caches in the local network for high-performance lookups.
5. *Switch-embedded structures:* We leverage the embedded `exact`, `range`, or `ternary`[5] match-action tables (using SRAM and TCAM) where we specify a set of flow identities to match and define the action as *allow*, *drop*, *rate_limit*.

[5]The term "ternary" refers to the memory's ability to store and query data using three different inputs: 0, 1, and wildcard.

In `ReportCtr` and `Recirculate`, we encapsulate the corresponding P4 `copy_to_cpu` and `recirculation` primitives with the packet header modifier.

Network Controller. The control part of Jaqen is implemented in Python and is connected to the switch control via RPC. This controller has three major functionalities: (a) Query statistics from the switch data planes and compute a detection logic via the detection API and Thrift API [74]; (b) Run or rerun the resource management heuristic by fetching current routing information and detection information from each switch as a global state; (b) Deploy mitigation modules via RPC to switch control and configure the stored mitigation modules via `switchd` daemon.

8 Evaluation

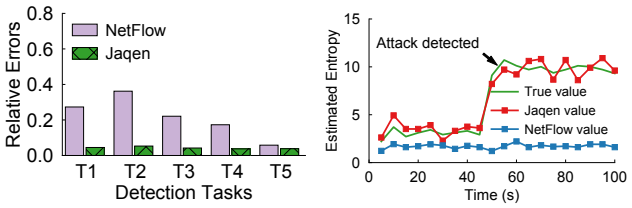
We evaluate Jaqen extensively on defending prevalent volumetric DDoS attacks [18] and demonstrate that:

1. Jaqen's obtains significantly more accurate metrics than out-of-band sampling approach. Jaqen's mitigation functions are more salable and effective than Poseidon [3].
2. Jaqen detects DDoS attacks with high accuracy and estimates the volumes of attacks with low errors ($< 3\%$).
3. Jaqen mitigates attacks with high effectiveness — low false positive and negative rates varying from 0.0 to 0.073.
4. Jaqen adapts to dynamic and variable-sized attacks within end-to-end 15 seconds with high effectiveness. Our larger scale network-wide simulator shows that Jaqen returns near-optimal resource allocation decisions within 1 sec.

Testbed. We deploy Jaqen on a testbed of one 6.5 Tbps Barefoot Tofino switch and eleven Dell R230 servers (Intel Xeon E2620 v4, 64GB RAM, 40Gbps Intel Network Interface Card). For single attack experiments, we use ten 40 Gbps servers to generate traffic and the remaining one as the targeted victim. We enable Intel DPDK [75] library on each server to achieve high-performance traffic generation. When sending legitimate traffic, we replay one-hour Internet traces from CAIDA [76] in a loop[6], at an aggregated packet rate of 59 Million packets per second (Mpps). For TCP related attacks, the sender maintains up to 1,048,576 legitimate TCP connections using virtual IP addresses. The controller of the switch pulls the detection result every 5 seconds.

Attack traffic generation. To evaluate Jaqen, we use real-world attack traces [77, 78] and launch a set of seven representative volumetric attacks: (1) To launch **SYN flood**, we use MoonGen [79] with DPDK [75] to send SYN requests with random source IP addresses. (2) To launch **ICMP flood** and **TCP/UDP elephant flows**, we implement custom Lua scripts to send ICMP, ACK/UDP traffic via MoonGen. (3) When launching **DNS and NTP amplification** attacks, we cannot exploit the public DNS and NTP servers to reflect the traffic. Instead, we set up local DNS and NTP servers forwarded to Google public DNS [80] and the server pool from ntp.org, and the local DNS responses will flood the victim servers. (4) To launch a **Memcached amplification attack**, we deploy Memcached [81] onto servers and send GET requests with forged source IPs to the targeted victim.

[6]When replaying traces, the TCP SYNs are sent without establishing actual connections.



(a) Errors in example detection tasks. (b) Entropy-based detection

Figure 11: Comparison with NetFlow [16].

Evaluation metrics and parameters. In estimating the attack volumes, we use *relative error* as $\frac{|detected\_vol - true\_vol|}{true\_vol}$, where $detected\_vol$ is the volume reported, and $true\_vol$ is the true volume of attack traffic. In the mitigation, we evaluate the false positive rate (FPR) and false negative rate (FNR). FPR is the rate of how much benign traffic is mistakenly mitigated as malicious traffic (false positives FP), defined as $\frac{FP}{FP+true\_negatives}$. FNR is the rate of how much malicious traffic is identified as benign traffic and is not mitigated (false negatives FN), defined as $\frac{FN}{FN+true\_positives}$.

For every period of time T (e.g., 5 sec in our experiments), the detection can be pulled by the controller to identify the occurrence, type, and volume of an attack. By default, if Jaqen detects the occurrence of the DDoS attacks for two consecutive time windows, the resource manager on the controller will compute the mitigation module allocation and deploy the needed modules to the switch. Thus, we measure the total reaction time as $|2 \times T + T_{Res\_allocation} + T_{Soft\_update}|$.

When configuring the probabilistic structures in the mitigation functions, the default blocked Bloom filters use 7 different hash functions with 44.04M entries (5.25MB memory). For DNS, NTP, and Memcached amplification attacks, the counting Bloom filters use 4 different hash functions with 11.01M entries (1.31MB memory) by default.

8.1 Comparison with Existing Solutions

Comparison with NetFlow on detection. NetFlow [16] is a standard network monitoring tool to conduct out-of-band network traffic analysis. Despite its processing delay in handling large batches of sampled packets, we compare the accuracy of five different DDoS-related detection tasks between Jaqen and NetFlow using two real-world attack traces [77, 78]. The tasks include T1: Unique source IPs, T2: Distinct 5-tuple flows, T3: Unique SYN connections, T4: Top sources in volume, and T5: Top victims in volume. We configure the sampling rate of NetFlow as 1/100 in order to keep up with the line-rate and low detection delay. Note that at this sampling rate, NetFlow stores a large number of packets and uses significantly

Defense (40G)	Poseidon (FPR / FNR)	Jaqen (FPR / FNR)
SYN proxy	2M, 25.2% / 1.3%	2M, 0.0% / 1.3%
DNS/NTP defense	2M, 1.2% / 3.7%	2M, 0.7% / 3.1%

Table 6: Jaqen vs. Poseidon on defense effectiveness.

more memory space than Jaqen. As shown in Figure 11(a), Jaqen's sketch-based detection has better accuracy than NetFlow across the tasks.

To evaluate entropy-based detection, we generate an HTTP flood attack. We manually inject source IPs from 50 randomly picked subnets and a range of victims that share a single 16-bit subnet. To launch the attack, we replay the Internet trace [76] with 70% probability to replace a packet with an attack packet. As presented in Figure 11(b), Jaqen captures the changes in the source IP entropy values and detects the occurrence of the attack, while NetFlow cannot accurately track the changes in the entropy values.

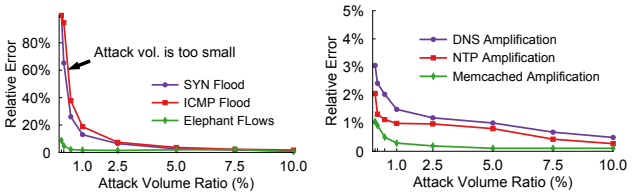
Comparison with Poseidon on mitigation. Poseidon[3] designs a hybrid DDoS mitigation solution with programmable switches and x86 servers. In handling 40Gbps volumetric attacks with 2M legitimate connections, both Poseidon and Jaqen can operate at the line-rate with μs-level latency, showing the promise of programmable switches. Further, we compare the mitigation FPRs/FNRs of the two approaches. With Poseidon [3], we configure their SYN proxy to use a large session table of size 2^{21} (8MB SRAM) while Jaqen uses 1.31MB SRAM. The monitor of Poseidon for DNS defense has a Count-Min sketch and we configure it to the same number of counters for both approaches. As reported in Table 6, Jaqen has a significantly better FPR (0% vs. 25.2%) than Poseidon in SYN proxy using 6× less memory. In handling DNS attacks, Jaqen's defense strategy (Figure 7) is more effective in reducing the FPR/FNR. We envision that Jaqen will support more sophisticated mitigation strategies to reduce the FPR/FNR for other attacks using the mitigation API.

8.2 Single Static Attack Evaluation

In this section, we perform experiments with the following conditions: (1) launch attacks with one attack method and different volumes; (2) deploy a single mitigation strategy to the switch; (3) legitimate clients maintain a reasonable number of normal connections with the targeted server.

Attack volume estimation. In Jaqen, the estimated volume of an attack is useful for the controller to compute the resource allocation decisions for mitigation. Our testbed can generate up to 380 Gbps traffic with different attack volumes. In general, a higher attack volume leads to a more accurate volume estimation as it becomes easier to catch. We generate different volumes of attack traffic for evaluation (0.1%, 0.2%, 0.5% to 10% of total).

As shown in Figure 12(a), when the flood attack volume is small from 0.1% to 0.2%, the volume estimation has large relative errors (from 134% to 32.9%). It is understandable



(a) Relative errors of estimating the volumes of flood attacks.

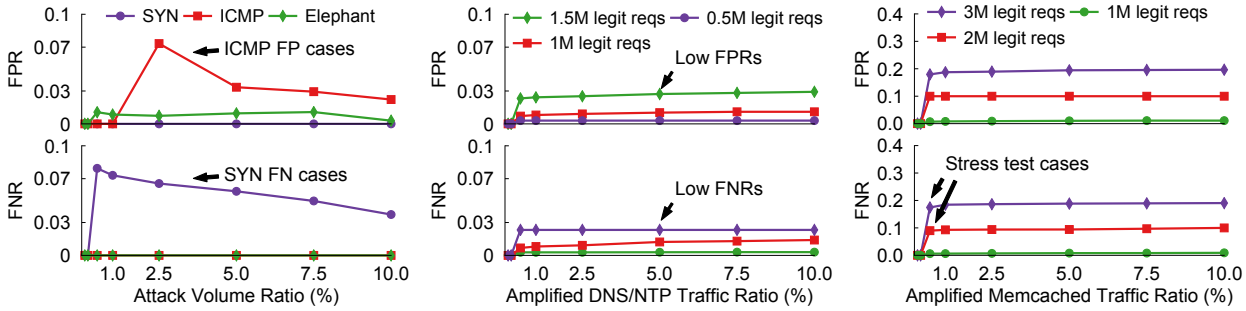
(b) Relative errors of estimating the volumes of amplification attacks.

Figure 12: Single static attack evaluation—part one.

as some legitimate traffic can be estimated as attack traffic at this attack volume. However, such attack traffic that is below our detection threshold 0.5%, poses small impacts to the network and technically has no difference from legitimate users. When attack traffic increases, the relative errors of the measured volumes significantly decrease to less than 2%. In addition, as depicted in Figure 12(b) for amplification attacks, the volume estimation incurs low relative errors (<3%) as most of the legitimate requests have received responses.

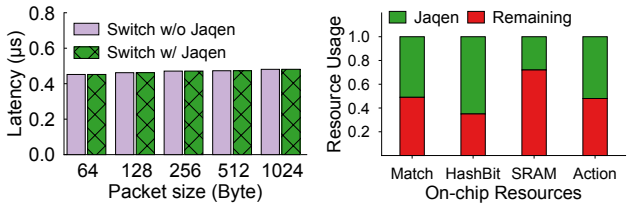
Mitigation effectiveness. To evaluate the effectiveness of Jaqen's mitigation, we conduct experiments leveraging six implemented attack countermeasures. As described in the following, each mitigation mechanism achieves low FPRs and FNRs when the attack traffic is significant:

- *SYN flood attack:* Our mitigation strategy has a switch-optimized SYN proxy and an approximate allowlist to record the legitimate flows passing the SYN cookie verification. There is some probability that the allowlist falsely allows some attack flows. This probabilistic error from Bloom filters incurs FNs for the mitigation as shown in Figure 13(a). As attack volume increases, the SYN-flood mitigation achieves 0 FPR and small FNRs.
- *ICMP flood attack:* We launch ICMP flood with generated ICMP Echo Request packets (`type_flag=8`) with random source IPs. ICMP packets are usually rare in the normal traffic and the best-practice is to block them completely [82]. Our reaction-based mitigation is better than that as it does not affect ping-based diagnosis in the normal case. As depicted in Figure 13(a), when the ICMP flood traffic is small from 0.1% to 0.5%, the counters will not raise the alarm as it is considered normal. When ICMP traffic is more significant, the filter starts blocking the ICMP Echo Requests which incurs some FPs.
- *Elephant TCP/UDP flows:* When the attacker leverages high-bandwidth zombie machines to launch a flood attack, they can generate elephant flows toward the targeted victim. We allocate different traffic bandwidths for the attack from 0.1% to 10% and split the bandwidth for 100 elephant UDP flows. We realize this attack by MoonGen rate limiting and a fixed-sized 100 random source IP generation when sending UDP packets. Figure 13(a) shows that the heavy hitter-based filtering achieves low FPRs and FNRs



(a) Mitigation FPR/FNR of on flood attacks. (b) Mitigation FPR/FNR on DNS/NTP attacks. (c) Mitigation FPR/FNR on Mem. attacks.

Figure 13: Single static attack evaluation—part two.



(a) Processing latency of a Barefoot switch w/ and w/o Jaqen. (b) Hardware resource of Jaqen with detection and mitigation modules.

Figure 14: Micro benchmarks.

from 0.009 to 0.012 as the underlying sketching algorithm guarantees high fidelity.

- *DNS/NTP amplification:* We set up local DNS servers with `BIND 9` [83]. On each local DNS server, we write a C++ custom packet generator to send forged DNS requests locally (≈ 0.9 Gbps per server) and the amplified responses (≈ 30 to 35 Gbps/server) are sent to the designated destination. Similarly, we set up local NTP servers with the NTP spool from ntp.org and generate modified NTP requests to the local NTP servers. As depicted in Figure 13(b), when there is 0.5 Mil legitimate requests inserted, the mitigation FPRs are negligible and FNRs are low from 0.022 to 0.029. While there are 1.5 Mil requests recorded, the FPRs and FNRs are still low from 0.024 to 0.028.
- *Memcached amplification:* In this attack, we conduct a stress test to add more numbers of unresponded Memcached requests to the allowlist. When the caching services are running abnormally with 2 to 3 Mil unmatched requests recorded in the CBF, the FPRs and FNRs will be increased to 0.1 and 0.18 (Figure 13(c)). Therefore, if there are indeed more benign flows to be recorded, more mitigation resources are needed.

8.3 Microbenchmarks

Latency: One advantage of using hardware switches for defense is that the processing latency is extremely small for legitimate traffic. To confirm this, we evaluate the processing latency of Jaqen with a detection module and three mitigation functions using different sized UDP packets sent from a single server with DPDK. As depicted in Figure 14(a), there is no noticeable processing latency change at the microsecond-level for a Jaqen-enabled switch.

Hardware resource usage: We measure the resource usage of Jaqen.[7] P4 allows developers to define their own packet formats and program the packet actions by a series of match-action tables, which are mapped into different stages in a sequential order, along with dedicated resources (e.g., match entries, hash bits, SRAMs, and action slots) for each stage. Figure 14(b) shows the resource usage of a switch with detection plus SYN, DNS, NTP mitigation modules. In effect, there is still adequate room for additional services.

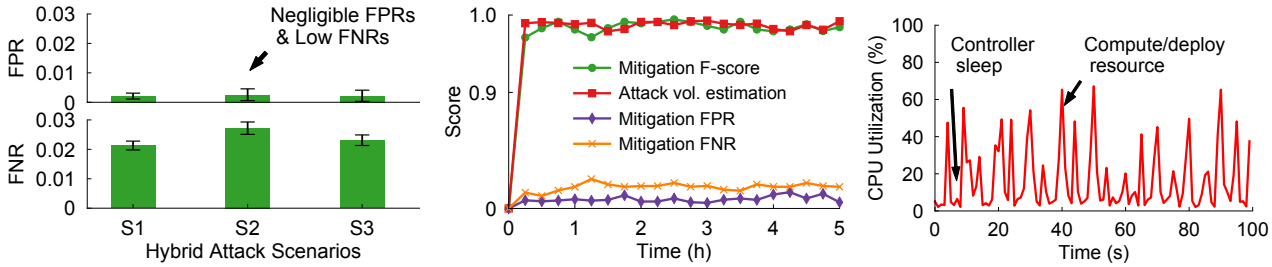
8.4 Large Hybrid and Dynamic Attacks

In this section, we evaluate the effectiveness of Jaqen when handling a hybrid of attack methods. Given that we have only one programmable switch available, we can deploy up to three different mitigation strategies at a time due to resource constraints. For simplicity, we consider four scenarios using three different attack methods using 90% of the total bandwidth (≈350 Gbps). For a hybrid and dynamic evaluation (S4), we run Jaqen for 5 hours with dynamically changing attacks (three randomly picked attack methods) every 15 min. For a larger-scale network-wide resource deployment, we evaluate using our greedy heuristic.

Scenario I [S1]: flood attacks. In this attack scenario, we launch three flood attacks simultaneously (SYN flood, ICMP flood, and elephant flows) with 120 Gbps each. As we can see in the first bar of Figure 15(a), Jaqen's mitigation modules work well with negligible FPR ($\leq$0.005) and median FNR=0.0213 in ten independent runs using legitimate traffic replayed from the CAIDA-2018 traces [76].

Scenario II [S2]: amplification attacks. In this attack scenario, we launch three amplification attacks simultaneously (DNS, NTP, and Memcached amplifications) with equal high throughput (120 Gbps). As presented in the second bar of Figure 15(a), Jaqen easily mitigates the majority of the attack traffic with tiny FPRs and small FNRs (≈0.0272).

[7] The actual switch hardware resources are proprietary information.



(a) FPR/FNR of mitigating different hybrid attacks. (b) 5-hour experiment on dynamic, hybrid attacks. (c) Controller CPU utilization on dynamic attacks.

Figure 15: Hybrid and dynamic attack evaluation.

Topologies	Nodes	Solver (s)	Jaqen (s)	Errors
Missouri Net.	67	113	0.03	0.0112
GTS CE	149	245	0.05	0.0161
Colt Telecom	153	273	0.05	0.0161
Dial Telecom	193	301	0.05	0.0162

Table 7: Jaqen vs. MIP solver on different topologies.

Scenario III [S3]: Flood and amplification attacks. In this attack scenario, we combine one flood attack with two amplification attacks (SYN flood, DNS amplification, and NTP amplification). Similar to the results of S1 and S2, Jaqen achieves ultra-low FPRs and FNRs.

Scenarios IV [S4]: Hybrid, dynamic, and variable-sized attacks. In this attack scenario, we launch a hybrid, dynamic attack with changing attack volumes. Every 15 min, we randomly pick three attacks from the total six methods with different volumes (each from 30 to 300 Gbps with 10 Gbps as an interval). As shown in Figure 15(b), Jaqen handles this scenario: Attack volume estimation has high accuracy $\geq$0.971, and the mitigation has high effectiveness shown in FPR, FNR, and F-score. On the controller side, we monitor the per-second CPU usage using Intel Vtune amplifier [84]. As depicted in Figure 15(c), large hybrid and dynamic attacks do not exhaust the controller CPU (Intel Pentium quad-core) due to our efficient offline estimation and resource allocation.

Network-wide simulation. To evaluate the efficiency and correctness of Jaqen's resource allocation algorithm, we pick four medium- to large-scale ISP topologies from Topology Zoo [71] and test them over our greedy heuristic simulator. As an ISP, we simulate BGP using Quagga [85] and Mininet [86], and randomly select ten edge routers with a valid configuration (all pairs routed). When simulating a 600 Gbps attack with six attacks, each BGP router will obtain valid routing decisions and these decisions are used as input to the resource manager. As in Table 7, the reported errors of our simulator are the relative errors from the optimal resource usage, and our resource managers can return a near-optimal allocation in real-time ($<$0.1 sec).

9 Other Related Work

In §2, 3, we have already discussed the closest related work. We cover other related work here and refer the readers to survey papers for further reading [87–89].

FPGA-based DDoS defense. Network vendors and researchers have proposed to build DDoS defense using FPGA such as [90, 91]. The research on exploring the flexibility of using FPGA for DDoS is limited [92–94]. As a general note, FPGA isn't as performant as programmable ASICs and has a much higher per Gbps cost.

NPU-based DDoS defense. NPU is one kind of network processors that target network applications, e.g., packet switching and firewalls. There are proposals on using NPU based hardware appliances to conduct DDoS defense and other security-oriented tasks, such as [95], [96].

Network telemetry using programmable switches. A number of recent works show how to use programmable switches for memory-efficient and per-packet level network telemetry [20, 57, 97–101]. These are related to our work focusing on DDoS detection and mitigation.

10 Conclusions

DDoS attacks remain a primary concern for Internet security today. The emerging programmable switches bring a unique opportunity to revisit ISP-scale DDoS defense for volumetric attacks. In this paper, we show that a performant, flexible, and cost-efficient ISP defense system is well within our reach. Jaqen leverages state-of-the-art switch-optimized strategies to achieve high detection accuracy and mitigation effectiveness, and Jaqen's attack coverage can be easily extended using the API. When handling large-scale attack volumes, Jaqen quickly reacts to dynamic and hybrid attacks with minimal latency for legitimate traffic. These demonstrated benefits, along with the natural high performance and low cost, make the programmable switch ASICs a viable challenger to the existing hardware appliances and software solutions in ISPs.

11 Acknowledgements

We would like to thank our shepherd Angelos Stavrou and the anonymous reviewers for their thorough comments and feedback that helped improve the paper. This work was supported in part by CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation program sponsored by DARPA, NSF Grants CNS-1700521, CNS-1813487, CNS-1955422, and CCF-1918757.

References

[1] "Q1 2018 DDoS Trends Report." https://bit.ly/2JDR1D9.

[2] "Transmission Control Protocol, DARPA Internet Program Protocol Specification." https://www.barefootnetworks.com/use-cases/in-nw-DDoS-detection/, Barefoot, 2019.

[3] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *Proc. of IEEE NDSS*, 2020.

[4] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *Proc. of IEEE Infocom*, 2017.

[5] J. Bai, J. Bi, M. Zhang, and G. Li, "Filtering spoofed ip traffic using switching asics," in *Proc. of ACM SIGCOMM Posters and Demos*, 2018.

[6] G. Grigoryan and Y. Liu, "Lamp: Prompt layer 7 attack mitigation with programmable data planes," in *Proc. of ANCS*, 2018.

[7] A. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection toprogrammable data planes," in *Proc. IFIP/IEEE IM*, 2019.

[8] "Arbor Networks APS Series." https://www.arbornetworks.com/ddos-protection-products/arbor-aps.

[9] "Cisco Guard XT 5650 Series." https://goo.gl/DoFRBk.

[10] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense," in *USENIX Security*, 2015.

[11] "Arbor Networks TMS Series." https://www.arbornetworks.com/ddos-protection-products/arbor-tms.

[12] "Barefoot Tofino." https://barefootnetworks.com/products/brief-tofino/.

[13] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, 2014.

[14] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering," in *Proc. of IEEE ICNP*, 2019.

[15] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. of ACM SIGCOMM*, 2002.

[16] "Cisco ios netflow." https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[17] M. Wang, B. Li, and Z. Li, "sflow: Towards resource-efficient and agile service federation in service overlay networks.," in *ICDCS*, pp. 628–635, IEEE Computer Society, 2004.

[18] "DDoS Breach Costs Rise to over $2M for Enterprises." https://goo.gl/o13QxD, Kaspersky Lab, 2018.

[19] V. Braverman and R. Ostrovsky, "Zero-one frequency laws," in *Proc. of STOC*, 2010.

[20] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proc. of ACM SIGCOMM*, 2016.

[21] C. Fachkha, E. Bou-Harb, and M. Debbabi, "Fingerprinting internet dns amplification ddos activities.," *CoRR*, vol. abs/1310.4216, 2013.

[22] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites," in *Proc. of WWW*, 2002.

[23] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "A fair solution to dns amplification attacks," in *Proc. WDFIA*, 2007.

[24] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *In Proc. of ACM SIGCOMM*, 2005.

[25] W. Lee and D. Xiang, "Information-theoretic measures for anomaly detection," in *Proc. of IEEE S&P*, 2001.

[26] H. Wang, D. Zhang, and K. G. Shin, "Detecting syn flooding attacks," in *In Proc. of INFOCOM*, 2002.

[27] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Proc. of USENIX NSDI*, 2013.

[28] "Intentional SYN Drop for mitigation against SYN flooding attacks." https://bit.ly/33S5eGf, 2018.

[29] A. Broder, M. Mitzenmacher, and A. B. I. M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Internet Mathematics*, 2002.

[30] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: The Count-min Sketch and Its Applications," *J. Algorithms*, 2005.

[31] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *ICALP*, 2002.

[32] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," in *European Symposium on Algorithms*, pp. 684–695, Springer, 2006.

[33] M. Tirmazi, R. Ben Basat, J. Gao, and M. Yu, "Cheetah: Accelerating database queries with switch pruning," in *Proc. of ACM SIGMOD*, 2020.

[34] "Arbor Networks APS Datasheet." https://www.netscout.com/sites/default/files/2018-04/DS_APS_EN.pdf, 2018.

[35] "NSFOCUS Anti-DDoS System Datasheet." https://nsfocusglobal.com/wp-content/uploads/2018/05/Anti-DDoS-Solution.pdf, 2018.

[36] "Stop DDoS Attacks before They Disrupt the Customer Experience." https://intel.ly/2N9hexa, 2020.

[37] "Where at&t keeps an all-seeing eye on its ginormous data-shuttling network." https://fortune.com/2016/04/30/att-gnoc-global-data-network-operations-center/, AT&T, 2016.

[38] "Edge-Core Networks - WEDGE100BF-65X-O-AC-F-US QSFP 100g." https://bit.ly/2HiZFW0.

[39] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proc. of ACM SIGCOMM*, 2017.

[40] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proc. of ACM SIGCOMM*, 2013.

[41] J. M. Smith and M. Schuchard, "Routing around congestion: Defeating ddos attacks and adverse network conditions via reactive BGP routing," in *Proc. of IEEE Symposium on Security and Privacy*, 2018.

[42] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "Senss against volumetric ddos attacks," in *Proc. of ACSAC*, 2018.

[43] "P4 Behavior Model version 2." https://github.com/p4lang/behavioral-model, 2018.

[44] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina, "Impact of packet sampling on anomaly detection metrics," in *Proc. of ACM IMC*, 2006.

[45] A. Ramachandran, S. Seetharaman, N. Feamster, and V. Vazirani, "Fast monitoring of traffic subpopulations," in *Proc. of IMC*, 2008.

[46] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. pf IEEE symposium on security and privacy*, 2013.

[47] "Multi-function Platform for Cloud Networking." https://bit.ly/2JhJQB6, Arista, 2018.

[48] "EX9200-Flexibility and scalability for business agility and growth." https://juni.pr/2JnC1tY, Juniper, 2018.

[49] "Google Cloud using P4Runtime to build smart networks." https://bit.ly/2Q7zG6B, Google, 2018.

[50] G. Antichi, T. Benson, N. Foster, F. M. V. Ramos, and J. Sherry, "Programmable Network Data Planes (Dagstuhl Seminar 19141)," *Dagstuhl Reports*, 2019.

[51] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, "An empirical evaluation of entropy-based traffic anomaly detection," in

Proc. of ACM IMC, 2008.

[52] P. Flajolet, ric Fusy, O. Gandouet, and et al., "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. AOFA*, 2007.

[53] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. of ACM IMC*, 2003.

[54] V. Braverman, R. Krauthgamer, and L. F. Yang, "Universal streaming of subset norms," *CoRR*, vol. abs/1812.00241, 2018.

[55] V. Braverman, R. Ostrovsky, and A. Roytman, "Zero-one laws for sliding windows and universal sketches," in *Proc. of APPROX/RANDOM*, 2015.

[56] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proc. of ACM ICM*, 2004.

[57] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. ACM SOSR*, 2017.

[58] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Proc. of RANDOM*, 2002.

[59] P. Flajolet, É. Fusy, O. Gandouet, and et al., "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *In Proc. of AOFA*, 2007.

[60] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *Proc. of SIGMETRICS/PERFORMANCE*, 2006.

[61] A. Chakrabarti, G. Cormode, and A. Mcgregor, "A near-optimal algorithm for estimating the entropy of a stream," *ACM Trans. Algorithms*, 2010.

[62] P. Clifford and I. Cosma, "A simple sketching algorithm for entropy estimation over streaming data," in *Proc. of AISTATS*, 2013.

[63] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: Analysing the rise of iot compromises," in *Proc. of USENIX WOOT*, 2015.

[64] M. Yang, J. Zhang, A. Gadre, Z. Liu, S. Kumar, and V. Sekar, "Joltik: enabling energy-efficient" future-proof" analytics on low-power wide-area networks," in *Proc. of ACM MobiCom*, 2020.

[65] Q. Xiao, Z. Tang, and S. Chen, "Universal online sketch for tracking heavy hitters and estimating moments of data streams," in *Proc. of IEEE INFOCOM*, 2020.

[66] Z. Liu, S. Zhou, O. Rottenstreich, V. Braverman, and J. Rexford, "Memory-efficient performance monitoring on programmable switches with lean algorithms," in *Proc. of SIAM APoCS*, 2020.

[67] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proc. ACM SOSP*, 2017.

[68] Z. Liu, Z. Bai, Z. Liu, X. Li, C. Kim, V. Braverman, X. Jin, and I. Stoica, "Distcache: Provable load balancing for large-scale storage systems with distributed caching," in *Proc. of USENIX FAST*, 2019.

[69] A. Zuquete, "Improving the functionality of syn cookies," in *Proc. IFIP TC6/TC11*, 2002.

[70] "Transmission Control Protocol, DARPA Internet Program Protocol Specification." https://tools.ietf.org/html/rfc793, DARPA, 1981.

[71] "The internet topology zoo." http://www.topology-zoo.org/.

[72] "Barefoot P4 Studio." https://www.barefootnetworks.com/products/brief-p4-studio/.

[73] "Jaqen Prototype Repo." https://github.com/Froot-NetSys/Jaqen, 2021.

[74] "Apache Thrift." https://thrift.apache.org/.

[75] "Data plane developer kit (dpdk)." https://software.intel.com/en-us/networking/dpdk.

[76] "The CAIDA UCSD Anonymized Internet Traces 2018." http://www.caida.org/data/passive/passive_2018_dataset.xml.

[77] "Capture Traces from Mid-Atlantic CCDC 2012." http://www.netresec.com/?page=MACCDC.

[78] "DARPA Scalable Network Monitoring (SNM) Program Traffic, Traces taken 2009-11-05 to 2009-11-05." https://www.impactcybertrust.org/dataset_view?idDataset=742.

[79] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proc. of IMC*, 2015.

[80] "Google Public DNS." https://developers.google.com/speed/public-dns/.

[81] "Memcached." https://memcached.org.

[82] "Ping flood (icmp flood)." https://www.imperva.com/learn/application-security/ping-icmp-flood/.

[83] "BIND 9 Open Source DNS Server." https://www.isc.org/downloads/bind/.

[84] "Intel vtune amplifier." https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[85] "Quagga routing suite." https://www.quagga.net/.

[86] "Mininet." http://mininet.org/.

[87] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, 2004.

[88] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: Classification and state-of-the-art," *Comput. Netw.*, 2004.

[89] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks.," *IEEE Communications Surveys and Tutorials*, 2013.

[90] "Fighting DDoS with Distributed Defense." https://bit.ly/2JGIkYG, 2016.

[91] C. Pham-Quoc, B. Nguyen, and T. N. Thinh, "Fpga-based multicore architecture for integrating multiple ddos defense mechanisms," *ACM SIGARCH Computer Architecture News*, 2017.

[92] N. Hoque, H. Kashyap, and D. Bhattacharyya, "Real-time ddos attack detection using fpga," *Comput. Commun.*, 2017.

[93] Y. Chen and K. Hwang, "Collaborative detection and filtering of shrew ddos attacks using spectral analysis," *J. Parallel Distrib. Comput.*, 2006.

[94] H. Chen, Y. Chen, and D. H. Summerville, "A survey on the application of fpgas for network infrastructure security," *IEEE Communications Surveys and Tutorials*, 2010.

[95] R. K. Thomas, B. L. Mark, T. Johnson, and J. Croall, "Netbouncer: Client-legitimacy-based high-performance ddos filtering," in *Proc. of DARPA DISCEX-III*, 2003.

[96] "Device, system and method for analysis of fragments in a fragment train." https://patents.google.com/patent/US20080127342, US Patent, 2007.

[97] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *Proc. of SOSR*, 2018.

[98] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *Deme session of ACM SIGCOMM*, 2015.

[99] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proc. of ACM SIGCOMM*, 2017.

[100] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proc. of ACM SIGCOMM*, 2019.

[101] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proc. of ACM SIGCOMM*, 2018.

Protocol	Attack	Description	Jaqen Mitigation Functions
TCP	SYN flood	Attackers send a large volume of fabricated SYN packets to exhuast victim servers' connection pools	`Block/AllowList()`, `ActionAndTest(drop,syn)` `HeaderHashAndTest(synproxy)`
	ACK flood	Attackers send forged ACK packets from diverse sources to the victim servers	`BlockList()`, `UnmatchAndAction(syn-ack,drop)`
	Elephant flows	Attackers send large TCP flows to exhaust victim's network bandwith	`Block/AllowList()`
	RST/FIN Flood	Attackers send fake RST or FIN packets to flood the victims and interrupt legitimate connections	`RateLimit()/BlockList()` `UnmatchAndAction(rst,drop)`
	DNS flood (TCP)	Attackers generate a high rate of DNS requests from different sources to exhaust DNS service.	`RateLimit()` `UnmatchAndAction(dns,drop)`
UDP	DNS amplification	Attackers launch forged DNS requests (with victim srcIPs) to public DNS resolvers; the replied traffic of an amplified volume will be directed to the victim	`RateLimit()/BlockList()` `UnmatchAndAction(dns,drop)`
	UDP flood	Attackers send a large volume of UDP packets from diverse sources to exhuast victim's bandwidth	`RateLimit()/BlockList()`
	NTP amplifaction	Attackers launch forged NTP requests (with victim srcIPs) to public NTP servers; the replied traffic of an amplified volume will be directed to the victim	`RateLimit()/BlockList()` `UnmatchAndAction(ntp,drop)`
	SNMP amplifcation	Attackers launch forged SNMP requests (with victim srcIPs) to SNMP servers; the replied traffic of an amplified volume will be directed to the victim	`RateLimit()/BlockList()` `UnmatchAndAction(snmp,drop)`
	SSDP amplifaction	Attackers forge the discovery requests with victim srcIPs to plug-and-play devices; the replied traffic of an amplified volume will be directed to the victim	`RateLimit()/BlockList()` `UnmatchAndAction(ssdp,drop)`
	Memcached amplification	Attackers discover open Memcached servers and send spoofed cache requests (with victim srcIPs) to flood the victim	`RateLimit()/BlockList()` `UnmatchAndAction(memcached,drop)`
	QUIC amplifaction	Attackers send spoofed "hello" messages to QUIC servers; the replied traffic of large volumes will be directed to the victim	`Block/AllowList()` `UnmatchAndAction(quic,drop)`
	DNS spoofing	Attackers send corrupt DNS records to volunerable DNS resolvers and poision the DNS cache	`KVStore(ip,record,65k)`
ICMP	ICMP flood	Attackers send a large volume of fabricated ICP echo requests from diverse sources	`RateLimit()/BlockList()`
	Smurf attack	A large number of spoofed ICMP echo requests with the intended victim srcIPs are broadcast to the network using an IP broadcast address	`Block/AllowList()` `RateLimit()`
ARP	ARP poisoning	Attackers send corrupt ARP mappings to a (local) network to cause denial of service or MITM	`KVStore(ip,mac,10k)`
Application layer	HTTP Get/Post flood	Attackers send a large volume of HTTP Get and Post requests to flood a target HTTP server	`BlockList()/RateLimit()`
	SIP register flood	Attackers try to send a high volume of SIP REGISTER or INVITE packets to SIP servers	`BlockList()/RateLimit()`
	Slowloris	Attackers launch a large number of small volume connections to exhaust victim server's connection pool	`Block/AllowList()` `RateLimit()`
	HTTP slow post	Attackers send many HTTP Post requests with message body in a slow rate to let the victim server time-out	`Block/AllowList()` `RateLimit()`

Figure 16: State-of-the-art volumetric attacks and their mitigation strategies in Jaqen.