SYN Flood Defense in Programmable Data Planes

Dominik Scholz, Sebastian Gallenmüller, Henning Stubbe, and Georg Carle
Technical University of Munich
{scholz|gallenmu|stubbe|carle}@net.in.tum.de

ABSTRACT

The SYN flood attack is a common attack strategy as part of Distributed Denial-of-Service, which steadily becomes more frequent and of higher volume. To defend against SYN floods, preventing valuable service downtime, malicious traffic has to be separated from legitimate TCP requests. For this challenge, sophisticated filtering mechanisms operating at high bandwidths are needed.

Modern programmable data plane devices can handle traffic in the 10 Gbit/s range without overloading. We discuss how we can harness their performance to defend entire networks against SYN flood attacks. Therefore, we analyze different defense strategies, SYN authentication and SYN cookie, and discuss implementation difficulties when ported to different target data planes: software, network processors, and FPGAs. We provide prototype implementations and performance figures for all three platforms.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network performance analysis**; **Middle boxes / network appliances**.

KEYWORDS

SYN flood mitigation, programmable data planes, P4

ACM Reference Format:
Dominik Scholz, Sebastian Gallenmüller, Henning Stubbe, and Georg Carle. 2020. SYN Flood Defense in Programmable Data Planes. In *3rd P4 Workshop in Europe (EuroP4'20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3426744.3431323

1 INTRODUCTION

The frequency, volume, and diversity of Distributed Denial-of-Service (DDoS) attacks continue to increase [17]. TCP SYN floods are a popular attack vector used in larger DDoS attacks [19, 20]. According to Kaspersky Lab's quarterly reports, from 2017 to 2020, the share of SYN flood traffic during large-scale DDoS attacks rose up to 92 %, becoming the "most popular type of attack" [11].

There are two potential mitigation methods against SYN flood attacks: *generic defense* that tackles any form of DDoS attack and *SYN-specific defense* fighting off TCP SYN flood in particular. Generic defense mechanisms remove, filter, or redirect malicious traffic through techniques such as blackholing of address ranges or per-flow tracking of traffic statistics [27]. The SYN-specific approach uses stateless client puzzles like SYN cookies or SYN authentication [2, 3]. These require the client to behave correctly beyond the initial SYN segment. The strength of the SYN-specific defense capabilities relies on the available performance to enforce and check correct TCP behavior before finishing a TCP handshake.

In this work, we investigate powerful off-the-shelf data planes such as the software-based DPDK, or programmable hardware such as Network Processing Units (NPU) or FPGAs that can be programmed using the P4 domain-specific language (DSL) to mitigate SYN flood attacks. Specifically, we focus on SYN-specific solutions that provide high service guarantees with low latency to legitimate clients while under SYN flood attack. We discuss the benefits and challenges of implementing these strategies using the P4 DSL compared to traditional software packet processing frameworks with kernel-bypass. We provide insights in regards to portability and target-specific code adaptions and use measurements to highlight connection success probability and latency.

The paper is outlined as follows. Section 2 summarizes the state of the art for SYN flood defense and deployment scenarios. Section 3 discusses the design decisions and challenges encountered in our prototype implementations using the P4 DSL. We compare our prototypes in regard to performance and resource consumption metrics in Section 4. Related work is presented in Section 5 before we conclude in Section 6. We provide additional details on SYN mitigation in a proxy setup (Appendix A), as well as a case study on SYN cookies in Linux (Appendix B).

2 SYN FLOOD MITIGATION

SYN flood mitigation can be separated into generic and SYN-specific approaches, deployed in different scenarios.

2.1 Generic Defense

The simplest mitigation approach is to blackhole all attack traffic, including SYN flood, during an ongoing attack, for instance, by filtering based on the source subnet. Unfortunately, this approach is also rejecting any legitimate connection attempts from these subnets. Another approach, using IP anycast to spread the load over multiple networks, increasing network resilience and the attack surface used to mitigate the attack, is an improvement [18]. However, an anycast network is challenging to implement [18] and during large attacks, collateral damage, impairing other services in the network, is possible [14]. Lastly, tracking per-flow statistics and using thresholds can be used to distinguish legitimate from suspected attack traffic [27].

The mentioned generic approaches are simple and effective, but highly unspecific. Using a shotgun approach, these techniques achieve their goal but can cause undesired effects, i.e., produce

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroP4'20, December 1, 2020, Barcelona, Spain
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8181-9/20/12…$15.00
https://doi.org/10.1145/3426744.3431323

false positives/negatives, therefore, reducing the service quality for legitimate clients.

2.2 SYN-specific Defense

Techniques like SYN cookie [3] or SYN authentication [16, 21] improve on the generic approaches. They have a narrow, highly specific focus; they employ sophisticated challenges for TCP clients to specifically target malicious SYN flood traffic. The widely deployed SYN cookie includes the challenge in the SYN/ACK, secured with a cryptographic hash that is bound to the flow, and expects an appropriate final segment of the TCP handshake from the client. Only when this is completed, the connection is forwarded to the application. SYN authentication whitelists the client or the whole subnet once the challenge is completed and then accepts future connections from this source. The challenge can either be to trigger a reset, react to a reset, finish the handshake, or initiate a higher-layer connection (e.g., HTTP request). Challenges can be combined with a cryptographic hash or fingerprinting techniques. We present more details regarding SYN cookies and SYN authentication in Appendix A. For the remainder of the paper, we focus on three techniques: regular SYN cookies, as well as SYN authentication performing a full handshake with and without including a cryptographic hash as cookie (referred to as $\text{Auth}_{\text{cookie}}$ and $\text{Auth}_{\text{full}}$, respectively). Only these strategies offer adequate protection while also achieving high service quality.

SYN authentication neither requires extensive memory nor CPU resources. The exception is the calculation of cryptographic hashes. Calculating cookie values is the limiting factor for efficiency, i.e., how much SYN flood can be processed. None of the SYN authentication strategies is transparent for the client application, as they reset the initial connection. A downside of SYN cookies is the limited support for TCP options.

SYN-specific strategies often compromise TCP signaling capabilities [21], as a retransmission of a SYN/ACK segment is not possible. This impairment is more severe for SYN cookies than for SYN authentication, as the latter works on the assumption that the client retries failed connection attempts. Assuming all packets are received correctly and within the timeframe of the calculated cookie, no technique classifies legitimate traffic as malicious. However, only SYN cookies and $\text{Auth}_{\text{cookie}}$ cannot be circumvented by malicious traffic due to the cryptographic client puzzle.

2.3 Deployment Scenario

Generic mitigation approaches like blackholing of volumetric attacks can be deployed anywhere in the network, preferably close towards the edge or as part of a traffic scrubbing center. The aforementioned SYN-specific strategies, targeting transport or application layer, are usually implemented directly on the server or using a separate node as proxy. When deployed *on the endhost*, mitigation does not scale as it only protects this particular node, and takes away the server's resources required to serve its regular purpose. Consequently, SYN flood mitigation is commonly deployed as *SYN proxy*. This proxy can be used to protect multiple servers, or, as part of a traffic scrubbing center or in the cloud, even multiple networks. We discuss the requirements for implementing a flexible and open-source SYN proxy using SYN cookies and SYN authentication.

SYN cookies, when used in a proxy setup (see Figure 6a in Appendix A), raise several issues. After finishing the initial handshake, the proxy cannot forward segments of the client to the server, as the server is still unaware of the connection. To uphold transparency, the proxy has to start a second connection between itself and the server. While the proxy can reuse the first handshake's sequence number in its SYN segment, the server will choose its own initial sequence number at random, i.e., it does not match the proxy's sequence number of the connection with the client. The proxy has to store the difference between these sequence numbers and modify sequence and acknowledgment numbers of all future segments of the connection.

Another issue is that the first data segment of the client is sent directly after finishing the first handshake. This poses a problem for the proxy, as its handshake with the server is likely not completed yet. If the proxy drops this data segment, the client retransmits it after a timeout, for which 200 ms is a common time period. Considering that typical RTTs are only a hundredth of this, this retransmission causes a significant delay penalty. A solution for the proxy is to temporarily store the client's initial data segment. Once the second handshake is completed, the stored segment can be translated and forwarded. Alternatively, the proxy can actively notify the client once the second handshake is complete, by resending the SYN/ACK segment, triggering a retransmission of the data segment. Further improvements include setting a window size of zero (*zero window*) in the original SYN/ACK segment, indicating that the server cannot process any more data. Until the SYN/ACK is resent with a non-zero window size, the client will not send the initial data segment, reducing bandwidth wasted for a segment that will be dropped.

SYN authentication is efficient due to its simplicity (see Figure 6b in Appendix A). The first connection attempt is interrupted. By whitelisting all further attempts for this client, the proxy does not have to keep a separate connection with the server or perform sequence number translation. No connection state has to be stored by the proxy, wherefore, a simple bitmap representing the whitelist is sufficient. However, as ACK segments of established connections cannot be distinguished from the third segment of the handshake, the proxy has to check every segment against the whitelist. If the origin is not whitelisted, the segment is assumed to be the third segment of the handshake and, when using $\text{Auth}_{\text{cookie}}$, the cookie hash is verified.

Comparison: SYN cookies and SYN authentication differ in regards to transparency and option support. SYN cookie has to modify every segment, while SYN authentication only modifies segments during the handshake. For the latter, state size depends on the whitelisting granularity, e.g., per flow or per subnet. However, state is reduced to a few bit per entry compared to keeping the sequence number difference per flow for SYN cookies. Both strategies need to perform a lookup for every segment (aside from SYN segments for SYN cookie) to determine the action. However, with P4 devices using match-action pipelines, this can be done efficiently.

Both approaches, SYN cookies and SYN authentication, violate the end-to-end principle of TCP, especially when deployed as proxy. However, *only* during attacks, where a TCP SYN flood would likely cause a service downtime, we deem the application of these approaches justified. Instead of no service, the goal is to provide a

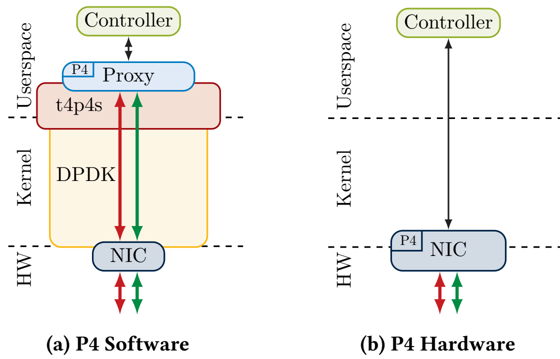


Figure 1: Architecture of SYN flood mitigation in programmable data planes

best-effort approach of service quality for legitimate flows, i.e., higher delay or minor connection disruptions are acceptable. We want to compare the different strategies; therefore, we focus our investigation on the tradeoff between efficiency (performance) and correct classification (false positives/negatives).

3 SYN PROXY IN P4

P4 [4] is a standardized DSL for programming software and hardware data planes. It enables rapid development cycles and creates portable implementations of network applications in software and hardware data planes. P4's match-action-based paradigm makes it an excellent language to realize packet filtering and DoS mitigation applications.

Opposed to raw packet handling, packets can only be modified through constructs allowed by the language. Furthermore, P4 has a clear separation between data and control plane (see Figure 1): while the data plane can match the entries of P4 tables to incoming packets and perform respective actions, only the control plane can insert new entries into a P4 table. The data plane communicates with the control plane, which in our case runs on the same node that also runs the P4 data plane, using digest messages.

3.1 Realized Programs and Targets

We implemented P4 programs for SYN cookie and SYN authentication strategies and tested their functionality using the Mininet-based bmv2 P4 switch target. For all platforms, the core P4 program, available as open source [22], remains the same. However, due to using different P4 architecture models and offering different extern interfaces, all platforms require small modifications to the P4 program. To reduce overhead, we only ported the implementations requiring simpler state maintenance, $\text{Auth}_{\text{full}}$ and $\text{Auth}_{\text{cookie}}$, to multiple P4 targets: t4p4s [26], a DPDK-based P4 software target running on commercial off-the-shelf (COTS) hardware (see Figure 1a); the NFP-4000 Agilio SmartNIC [10] NPU (see Figure 1b); and the NetFPGA SUME [9].

3.2 Program Core

The P4 implementations for the mentioned strategies follow the same structure independent of the used strategy. At first, packets are parsed up to and including the TCP header. The following match-action pipeline at its core works as an L2 forwarder. Depending on the determined outgoing port, MAC addresses are updated using a table lookup. As P4 cannot generate new packets, the received packet is modified according to the strategy used, TCP flags set in the received packet, and the state kept by the proxy. State—whitelist or sequence number difference—is maintained as match-action table, requiring one lookup for every segment. No changes to the IP layer are performed besides exchanging IP addresses, requiring only an update of the TCP checksum before the packet is transmitted.

3.3 Target-specific Changes

Aside from the core logic that is portable between different P4 targets, two aspects require target-specific changes. First, individual targets use different P4 architecture models, i.e., the order or number of pipeline stages and the extern interfaces differ. For the former, this only requires changes to the structure of the program depending on the concrete model (e.g., v1model for t4p4s and SimpleSumeSwitch for NetFPGA). The extern interfaces are challenging for SYN mitigation as it requires a cryptographic hash function as extern. For a software target like t4p4s, or a target that can be extended using software like the NPU, even complex externs can be tightly integrated by the developer. For other, in particular, hardware targets, other approaches like adding the functionality as separate pipeline block, extending the architecture model, is possible [23]. If externs cannot be changed or added (e.g., ASIC), functionality can be offloaded to other hardware accelerators or a separate node. Even if the main computational task of creating a SYN cookie, the hash calculation, is not part of the P4 language, P4 is beneficial as the processing logic can be easily implemented in a portable and understandable fashion.

3.4 Cookie Calculation

To calculate a standard-conform cookie, the P4 target needs to offer functionality for generating a timestamp (replay protection) and hash calculation. Integration of cryptographic hash functions in P4 data planes is currently possible for software, NPU, and FPGA targets [23]. The integration and use of, for instance, SipHash as extern on the software target is straight forward as it can be added as library to the hardware-dependent t4p4s code. We choose SipHash as it is designed for short inputs like packet data [1], while achieving good performance when integrated into P4 targets [23]. Although the NPU target includes a crypto accelerator for SHA1 and SHA2, it was unavailable on our card, wherefore we opted to integrate a SipHash function as extern. The NPU allows to add extern functions written in a variant of C used to program its processing cores. We are not aware of a P4 ASIC that supports cryptographic functions.

Replay protection can also be achieved by using a table containing a counter, which is updated by the control plane.

3.5 Whitelisting

The easiest approach to perform whitelisting in a P4 program is to use a match-action table. The data plane informs the control plane through a digest message whenever a flow or IP address should be whitelisted and the control plane inserts an according table entry. The disadvantage is that this communication results in delay until the rule takes effect in the data plane.

An alternative approach that does not include a control plane is to use a Bloom filter data structure built with registers. This precludes additional communication delays, however, depending on the target and implementation of the register extern, registers might require more resources or have slower read/write access times compared to match-action table entries. Furthermore, additional complexity is required to maintain the state in the Bloom filter, in particular, evicting outdated entries. As both methods have tradeoffs, which method to use depends on the concrete target device and deployment requirements, e.g., if the delay to insert table entries is acceptable.

3.6 Buffering Packets

Stalling the initial TCP data segment when using the SYN cookie strategy is not possible with P4, as P4 has no construct to write entire packets to memory. Stalling is only possible, if the target provides an extern for this task. To perform this operation at line-rate in a programmable ASIC, FPGA, or NPU, however, is unlikely due to the memory capacity and memory bandwidth required.

An alternative is to use a secondary COTS device as storage server (*slow-path*), programmed using, for instance, a framework like DPDK. If a packet needs to be stalled, the proxy forwards this segment to the storage server. Once the handshake is finished, the proxy informs the storage server, e.g., via the controller, to transmit the stalled packet. The downsides are the increased complexity for the hardware setup, as well as necessary controller logic.

As the underlying problem can be circumvented by using a TCP zero window or active notification by resending the SYN/ACK (cf. Section 2.3), we did not implement this slow-path solution.

4 EVALUATION

Using a standardized DSL like P4 makes program development simpler and portable to ASIC, FPGA, and NPU devices, but comes at the cost of flexibility as the set of functions offered is limited. The following uses empirical measurements to compare performance indicators of the discussed implementations. We also evaluate a SYN proxy developed by us with the software packet processing framework libmoon [8] based on DPDK [22]. As our implementation uses the same underlying framework as t4p4s, we can compare the P4 solution to a complex implementation using raw packet handling and kernel-bypass techniques.

4.1 Key Performance Indicators

The primary performance indicator for a SYN proxy is the total SYN flood processed. From a user perspective, e.g., the number of HTTP requests served without packet loss and the overall latency are a concern. While, in general, the latency of a device or application when operating in an overload scenario is not of interest, in the case of a SYN proxy, it is highly likely that the proxy will reach an overload state during a high volume attack. We, therefore, analyze latency values in low (no SYN flood), middle (50 % of total processed SYN flood), and overload scenarios.

4.2 Measurement Setup

As shown in Figure 2, the load generating host sends malicious SYN flood and legitimate HTTP traffic via two separate links to the Device-under-Test (DuT). Using a 10 GbE switch, the traffic is mixed so that malicious and legitimate traffic arrive at the DuT at the same port, i.e., are indistinguishable based on the ingress port. We use MoonGen [6] as load generator for the SYN flood traffic. A constant load of HTTP queries is generated using wrk2 [25]. All measurements are run for 30 s, allowing for accurate latency results up to the 99.99 %-ile. The DuT forwards traffic classified as legitimate to a separate host hosting the webserver.

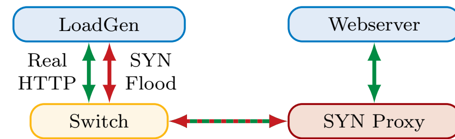


Figure 2: Measurement Setup

For measurements using software targets, the DuT is a COTS server, equipped with an Intel X722 NIC and an Intel Xeon Gold 6130 CPU clocked at 2.1 GHz. In other scenarios, the DuT is a server equipped with either a 10 GbE P4-programmable NPU or FPGA. For all measurements with the COTS system, we disabled turbo boost, set the CPU frequency to the maximum of 2.1 GHz, and pinned all traffic to one CPU core. The webserver (nginx, v1.10.3) is limited to a single worker and CPU core and serves a 1 kB static website.

4.3 Webserver Overload

Preliminary tests show that the webserver is capable of processing up to 30 000 HTTP requests per second for 10 to 10 000 parallel connections. However, latency increases when using more than 1 000 parallel connections or more than 4 000 requests per second. We measured a reduced connection probability for more than 700 parallel connections, even when issuing only 100 HTTP requests per second. As we do not want to measure overload artifacts of the webserver, we restrict our measurements to 100 parallel connections, issuing a total of 100 or 1 000 HTTP requests per second.

4.4 Processed SYN Flood

Figure 3 shows the maximum SYN flood each implementation is able to process. For all implementations, the use of a cryptographic hash function is the limiting factor and reduces the maximum processed SYN flood by up to 50 %, which is comparable to other studies [23]. Due to the possibility to manually optimize and parallelize packet processing, the libmoon/DPDK implementation achieves up to 50 % better performance than the t4p4s implementation using P4. In contrast, the libmoon/DPDK implementation requires approx. 1 000 lines of code and careful development and optimization. Only the hardware P4 targets are capable of processing up to 14 Mpps of SYN flood traffic when using the simpler $\text{Auth}_{\text{full}}$ strategy.

All implemented strategies scale linearly with the number of cores or devices used (not shown), such that even when cookies are calculated, throughput close to line-rate can be reached.

4.5 Quality of HTTP Requests

For all implemented solutions the connection probability for 100 HTTP requests per second is at 100 % until the point of overload (Figure 4). After this point, the probability slowly drops. As the webserver is not overloaded, all requests reaching the server are

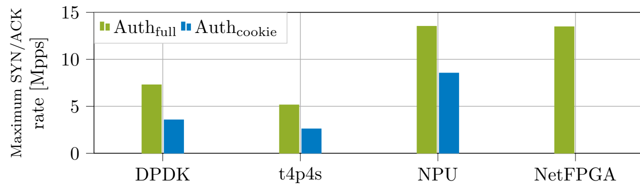


Figure 3: Maximum processed SYN flood traffic

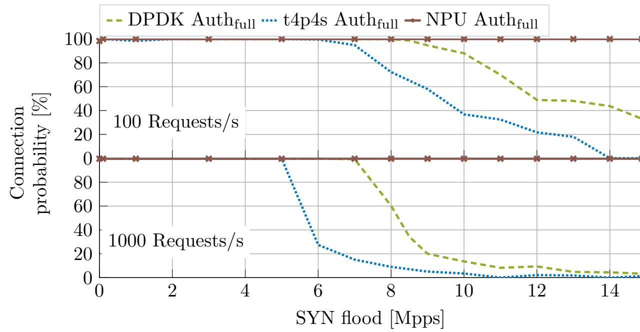


Figure 4: Connection probability for 100 and 1 000 HTTP requests per second

served. However, with increasing SYN flood, causing processing overload for the proxy, the chance that the proxy is able to process and forward legitimate traffic drops. As the NPU is able to process the SYN flood at almost line-rate, no HTTP packet is lost.

More requests per second reduce the connection probability during overload. This is due to the same number of parallel connections being used. The number of connections experiencing a timeout remains the same, however, in the case of having more requests per second for a given connection, one timeout has a larger impact. Increasing the number of parallel connections reverts this effect.

To reduce clutter, we do not show $\text{Auth}_{\text{cookie}}$. For these strategies, the slope is the same as for their respective $\text{Auth}_{\text{full}}$ counterparts, however, shifted to the left. This shift is correlated to the reduced maximum SYN flood that can be processed. For the NPU platform the probability starts dropping when reaching approximately 10 Mpps.

Connection latencies for the best case (no SYN flood), average case (50 % SYN flood, relative to maximum processed flood) and worst case (overload, maximum processed flood) are shown in Figure 5. For most scenarios, the median latency is between 1 and 1.4 ms, while for the no flood and 50 % cases a long-tail up to 4 ms is visible. Both implementations for the CPU target show sporadic outliers and a long-tail behavior with up to a second already for the 90th percentile during overload. The long-tail is expected due to batch processing and operating system interrupts typical for software packet processing frameworks like DPDK [7]. Due to the lower probability when issuing 1 000 HTTP requests per second for 100 parallel connections, the median latency during overload is above one second.

The exception is the NPU, showing latencies between 1 and 4 ms without outliers even under overload. Further, the latency for no flood is worse compared to when increasing the SYN flood

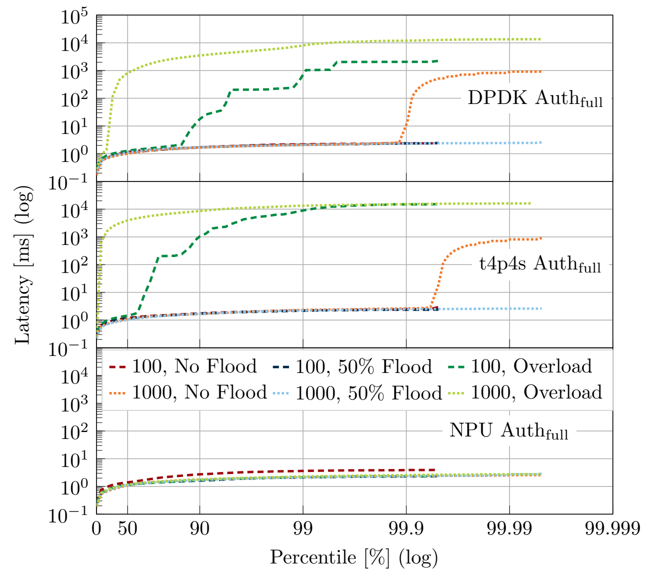


Figure 5: High dynamic range latency for 100 and 1 000 HTTP requests per second

load. We attribute this to specifics of the architecture, improving processing when increasing the backpressure on internal buffers or when reducing idle cycles caused by energy-saving features.

For the $\text{Auth}_{\text{full}}$ implementation on the NetFPGA SUME, the latency for up to 1 000 HTTP requests per second was stable between 1 to 4 ms without outliers up to a SYN flood of 1.5 Mpps (not shown). However, for reasons unknown to us, the program stopped working for higher loads of mixed traffic. Related work shows that the NetFPGA SUME, in general, is able to run P4 programs of higher complexity with latency below 10 µs and no long-tail. This is even possible when modifying the P4 architecture of the NetFPGA to integrate a SipHash or SHA3 core, which can be used to hash even complete packet data. [23]

4.6 Whitelisting — Scaling Match-Action Tables

Our previous work [24] shows that the number of entries in match-action tables on P4 targets scale in regards to performance and resource usage. For instance, more than one million 32 B exact match entries for the t4p4s DPDK can be inserted. On a software target, the bottleneck becomes the CPU cache, quickly reducing the performance when exceeding L3 cache capacity. However, adding even more entries is still possible. On a hardware target, exhausting the available resources is a hard limit. While one entry depends on its match size (i.e., the key), action data, and action performed, sufficient resources for up to several hundreds of thousands of entries are available on, for instance, the commercial Intel Tofino ASIC (neglecting resources required by the rest of the P4 program). [24]

If the resources for whitelisting are restricted by the P4 target, whitelisting can also be aggregated. Instead of whitelisting individual flows identified by the 5-tuple, whitelisting can be performed based only on the source IP or even the complete source subnet. This results in a tradeoff between whitelisting granularity and resource consumption.

4.7 FPGA Resource Consumption

For the NetFPGA SUME, the synthesized P4 proxy program only uses up to one third of the total resources. This leaves enough resources to further enhance the program to defend against other attacks or to add a SipHash implementation into the P4 pipeline [23], allowing to even perform $\text{Auth}_{\text{cookie}}$.

4.8 Programming Experience

Implementing SYN mitigation mechanisms in programmable data planes is drastically simplified using the P4 language compared to raw packet handling using, for instance, libmoon/DPDK. Its well-defined programming language, architecture model and extern interfaces aid the implementation of the overall logic independent of the target platform, without the need to take care of memory and buffer management. Furthermore, P4 supports hardware targets, where implementing such complex processing was previously infeasible or required domain-specific knowledge. However, depending on the concrete target, it is subjective and domain-specific influences of the target device can cause challenges. For instance, when using t4p4s, knowledge about the code architecture, DPDK and C is required, while debugging the NetFPGA SUME requires VHDL expertise. All targets provide different programming frontends, ranging from Linux command-line scripts for t4p4s and the P4→NetFPGA toolchain, to programmer studios for the NPU. Thereby, an expected difference in the user experience can be noted between research projects and commercial products.

5 RELATED WORK

Besides Linux, other major operating systems like Windows and FreeBSD utilize TCP SYN cookies as preferred mitigation method, enabled by default during periods with high traffic volumes [13]. Cf. Appendix B for a performance case study of Linux SYN cookies.

SYN cookies, as part of a SYN proxy, have been implemented by multiple projects. A *SYNPROXY* module for the netfilter framework is available in Linux [12]. Managing the SYN flood, *SYNPROXY* forwards only legitimate traffic to the Linux kernel. To do so, the initial SYN segment is intercepted by netfilter, calculating a SYN cookie. Once the client finishes connection establishment with a verified cookie, the proxy sends a SYN segment to the original server destination, using the initially negotiated options. After finishing the second handshake, the proxy is only involved in sequence number and timestamp translation [12]. This approach enables mitigation of a 2 Mpps SYN flood using only 7 % CPU utilization on an eight core test system [12]. The disadvantage is that it is integrated with a Linux end-host, i.e., it cannot be deployed as proxy or ported to different target platforms. Our P4 implementations improve on this as they can be deployed on any P4 target in a proxy or end-host setup.

Zhang et al. propose Poseidon [27], a DDoS defense framework that maps customizable mitigation strategies to programmable data planes in the network. The use-case is as part of a scrubbing center, cleaning the traffic from not only SYN flood traffic, but general DoS traffic of customer networks. Poseidon uses generic defenses like packet counting and probabilities to mitigate most attack vectors, including SYN floods. For HTTP floods, Poseidon does not create client puzzles in P4, but creates them in a separate DPDK proxy. This comes at the cost of higher end-to-end latency (approx. 70 ms), similar to existing commercial solutions. We improve on Poseidon's SYN flood mitigation by performing puzzles in the data plane without the need for probabilities. In fact, our solution could be integrated as part of Poseidon's network orchestration framework.

Large-scale commercial solutions exist, however, due to the closed-source nature, implementation details are rare. For instance, the Arbor Threat Mitigation System [15] provides middleboxes of different scale for traffic scrubbing, including unspecified SYN flood mitigation. Cloudflare offloads the TCP handshake to the cloud using an IP anycast network [5, 18]. Only once the handshake completes it is forwarded to the target server.

6 CONCLUSION

SYN floods are still the predominant traffic for high-volume (D)DoS attacks on the Internet. The client puzzle—including a cryptographic hash value—as part of SYN cookies or SYN authentication is the single effective SYN-specific defense strategy that is not based on blackholing or heuristics. It guarantees that no malicious connection attempts are successful, while not falsely rejecting legitimate requests. Due to the computational complexity of cryptographic hash functions, this is the bottleneck for these implementations. The SYN cookies and SYN authentication strategies both offer similar protection capabilities, with the former being fully transparent for TCP clients and the latter being simpler to implement.

Our programmable data plane prototypes have shown that SYN authentication, when used in a proxy setup, can mitigate SYN floods at 10 GbE line-rate. The P4 solutions are easy to implement and can be ported to different target platforms, in particular, hardware devices, which achieve end-to-end connection latencies below 5 ms with low jitter. The P4 software target achieves latencies comparable to implementations using the libmoon software packet processing framework.

We conclude that effective and efficient SYN flood mitigation on modern data planes is possible. With both mitigation strategies, SYN cookies and SYN authentication, performing equally well, we recommend SYN authentication, being the simpler one to implement. The crucial limiting factor for hardware data plane solutions is the availability of a suitable cryptographic hash function. However, cryptographic hash operations can be implemented in hardware efficiently—demonstrated by large-scale experimental prototypes such as Bitcoin—which would allow for powerful data plane driven SYN flood mitigation.

AVAILABILITY

The source code of our SYN proxy implementations for libmoon and the bmv2 P4 target is available at [22].

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (project ModANet under grant no. CA595/11-1) and the German-French Academy for the Industry of the Future. The authors want to thank Bassam Jaber for his valuable contributions to the P4 implementations in bmv2, Minoo Rouhi for her survey on SYN proxy mitigation methods, and the anonymous reviewers for their valuable feedback.

REFERENCES

[1] Jean-Philippe Aumasson and Daniel J Bernstein. 2012. SipHash: a fast short-input PRF. In *International Conference on Cryptology in India*. Springer.

[2] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. 2000. DOS-resistant authentication with client puzzles. In *International workshop on security protocols*. Springer.

[3] D. J. Bernstein. 1996. SYN cookies. [Online]. Last visited 2020-09-07. Available: http://cr.yp.to/syncookies.html.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* (2014).

[5] Cloudflare. 2011. SYN Flood Attack. [Online]. Last visited 2020-10-13. Available: https://www.cloudflare.com/en-gb/learning/ddos/syn-flood-ddos-attack/.

[6] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. ACM.

[7] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. 2015. Comparison of Frameworks for High-Performance Packet IO. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2015)*. Oakland, CA, USA.

[8] Sebastian Gallenmüller, Dominik Scholz, Florian Wohlfart, Quirin Scheitle, Paul Emmerich, and Georg Carle. 2018. High-Performance Packet Processing and Measurements (Invited Paper). In *10th International Conference on Communication Systems & Networks (COMSNETS 2018)*. Bangalore, India.

[9] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4->NetFPGA Workflow for Line-Rate Packet Processing. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[10] Netronome Systems Inc. 2016. *NFP-4000 Theory of Operation*. Technical Report. https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf Last accessed: 2020-09-07.

[11] Oleg Kupreev, Ekaterina Badovskaya, and Alexander Gutnikov. 2020. DDoS attacks in Q1 2020. [Online]. Last visited 2020-8-26. Available: https://securelist.com/ddos-attacks-in-q1-2020/96837/.

[12] Patrick McHardy. 2013. netfilter: implement netfilter SYN proxy. [Online]. Last visited 2020-09-07. Available: https://lwn.net/Articles/563151/.

[13] Microsoft TechNet. [n.d.]. Syn attack protection on Windows Vista, Windows 2008, Windows 7, Windows 2008 R2, Windows 8/8.1, Windows 2012 and Windows 2012 R2. [Online]. Last visited 2020-09-07. Available: https://blogs.technet.microsoft.com/nettracer/2010/06/01/syn-attack-protection-on-windows-vista-windows-2008-windows-7-windows-2008-r2-windows-88-1-windows-2012-and-windows-2012-r2/.

[14] Giovane CM Moura, Ricardo de O Schmidt, John Heidemann, Wouter B de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. 2016. Anycast vs. DDoS: Evaluating the November 2015 root DNS event. In *Proceedings of the 2016 Internet Measurement Conference*. 255–270.

[15] Netscout Systems, Inc. 2018. Arbor Threat Mitigation System (TMS). Data Sheet [Online]. Last visited 2020-10-13. Available: https://www.netscout.com/product/arbor-threat-mitigation-system.

[16] Raluca Oncioiu and Emil Simion. 2018. Approach to Prevent SYN Flood DoS Attacks in Cloud. In *2018 International Conference on Communications (COMM)*. IEEE.

[17] Charlie Osborne. 2020. 16 DDoS attacks take place every 60 seconds, rates reach 622 Gbps. [Online]. Last visited 2020-8-26. Available: https://www.zdnet.com/article/16-ddos-attacks-take-place-every-60-seconds-rates-reach-622-gbps/.

[18] Prince, Matthew. 2011. A Brief Primer on Anycast. The Cloudflare Blog [Online]. Last visited 2020-10-13. Available: https://blog.cloudflare.com/a-brief-anycast-primer/.

[19] radware Inc. 2013. DoS Cyber Attack Campaign Against Israeli Targets. [Online]. Last visited 2020-09-07. Available: https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/dos-cyber-campaign-against-israeli-targets/.

[20] radware Inc. 2013. Operation Ababil. [Online]. Last visited 2020-09-07. Available: https://security.radware.com/WorkArea/DownloadAsset.aspx?id=848.

[21] Livio Ricciulli, Patrick Lincoln, and Pankaj Kakkar. 1999. TCP SYN flooding defense. CNDS.

[22] Dominik Scholz and Bassam Jaber. 2020. SYN Proxy Implementations. [Online]. Last visited 2020-09-07. Available: https://github.com/syn-proxy.

[23] Dominik Scholz, Andreas Oeldemann, Fabien Geyer, Sebastian Gallenmüller, Henning Stubbe, Thomas Wild, Andreas Herkersdorf, and Georg Carle. 2019. Cryptographic Hashing in P4 Data Planes. In *2nd P4 Workshop in Europe (EUROP4)*. Cambridge, UK.

[24] Dominik Scholz, Henning Stubbe, Sebastian Gallenmüller, and Georg Carle. 2020. Key Properties of Programmable Data Plane Targets. In *Teletraffic Congress (ITC 32), 2020 32nd International*. Osaka, Japan.

[25] Gil Tene. 2012. wrk2 - a HTTP benchmarking tool based mostly on wrk. [Online]. Last visited 2020-09-07. Available: https://github.com/giltene/wrk2.

[26] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. „T4P4S: A Target-independent Compiler for Protocolindependent Packet Processors". In *IEEE HPSR*.

[27] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *Proceedings of NDSS*.

A SYN PROXY MITIGATION PROCEDURES

In the following, we explain the actions performed by a SYN proxy when deploying SYN cookie or SYN authentication strategies.

A.1 SYN Cookie

For SYN cookies [3], state, usually kept by the server, is encoded and put into the initial server-side sequence number (y in Figure 6a). A legitimate client will finish the handshake sending a TCP ACK segment, setting the acknowledgment number to $y + 1$. The server only accepts the connection and creates the necessary state, if the received number can be decoded and verified. Using standard SYN cookies, the 32 bit initial sequence number is made up of three different values: a timestamp value to prevent the collection and re-injection of old cookies [3]; the Maximum Segment Size option as it is essential for TCP performance; and a cryptographic hash of the connection 4-tuple (source and destination IP addresses and ports) and the timestamp value. This way, it is infeasible for an attacker to create a valid SYN cookie by itself.

Only after the verification of the cookie, the full state object for an established TCP connection is created by the proxy. As the server is unaware of the connection, the proxy has to perform a second handshake with the server. The server chooses a different initial sequence number (z), wherefore the proxy has to store the difference between the client- and server-side number (δ). For all subsequent segments of the connection, the proxy has to modify sequence and acknowledgment numbers accordingly.

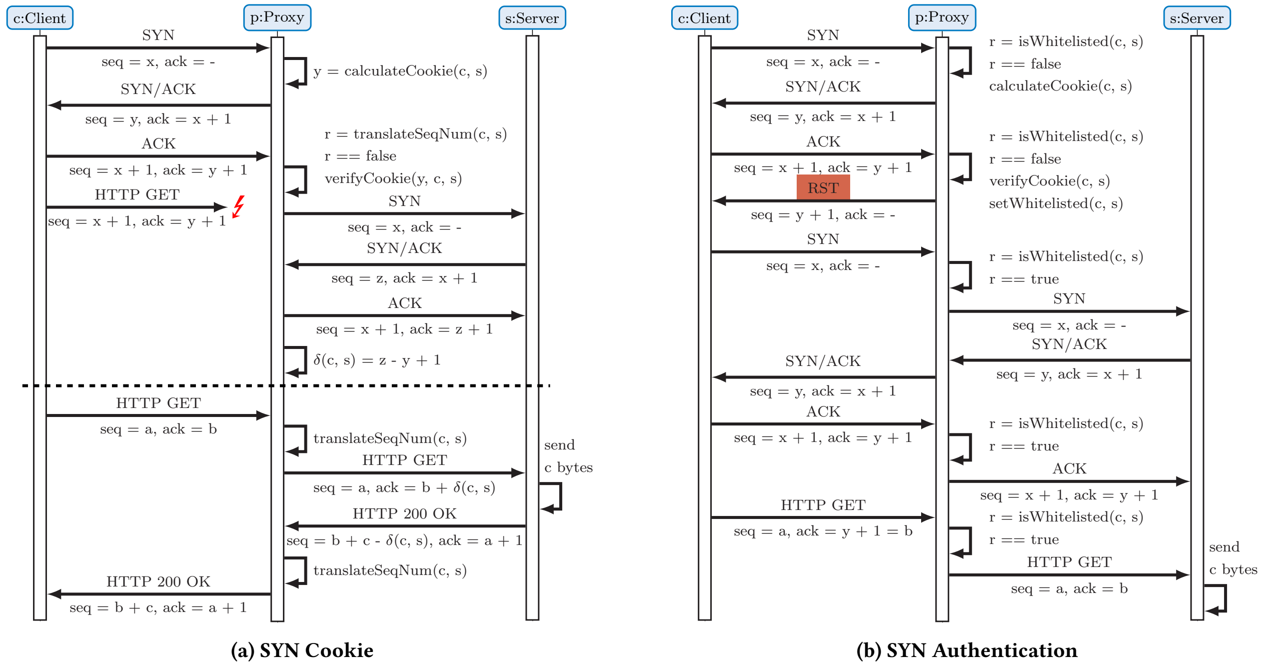
A.2 SYN Authentication

SYN authentication aims to further reduce the resources required to verify the client's legitimacy. Figure 6b shows SYN authentication using a full TCP handshake with cryptographic cookie ($\text{Auth}_{\text{cookie}}$). Once the client finishes the handshake, i.e., the client is willing to create its own TCP connection state, the server resets the connection and whitelists future connection attempts. These attempts are then directly forwarded by the proxy, requiring no additional TCP handshake with the server or modification of the packets.

$\text{Auth}_{\text{full}}$ is the same procedure as $\text{Auth}_{\text{cookie}}$, however, skipping the hash calculation and verification. Consequently, $\text{Auth}_{\text{full}}$ is easier to circumvent.

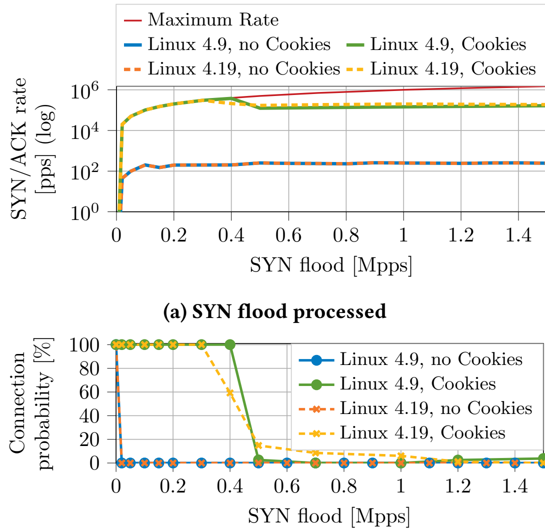
B CASE STUDY: SYN COOKIES IN LINUX

The Linux TCP/IP stack has SYN cookies enabled by default, but only uses them when the backlog, the buffer for unfinished TCP connections, of a socket is already full. As a case study, we analyze two different Linux versions when subjected to a SYN flood, while serving a static website: 5.9.0 and 4.19.0, using SHA1 and SipHash as cookie hash function, respectively. Figure 7 shows the amount of processed SYN flood and the probability of successfully serving 100



(a) SYN Cookie **(b) SYN Authentication**

Figure 6: Message Exchange for SYN flood mitigation strategies



(a) SYN flood processed

(b) HTTP requests served (100 requests per second offered)

Figure 7: Performance of Linux during SYN flood

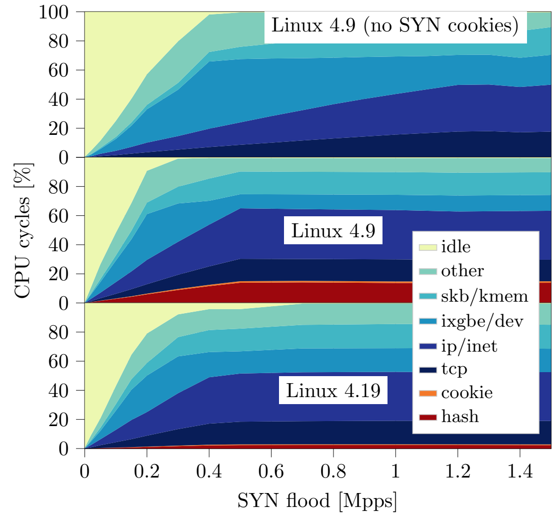


Figure 8: Profiling of Linux during SYN flood

HTTP requests per second for an increasing SYN flood on a single CPU core. When disabling SYN cookies, Linux can only process up to 250 SYN packets per second. However, no HTTP requests are served. Profiling (see Figure 8) reveals that this behavior is not due to CPU exhaustion. Instead, the TCP backlog is the limiting factor.

When enabling SYN cookies, both Linux versions behave similarly, i.e., process up to 0.4 Mpps of SYN flood, while all HTTP requests are served. When further increasing the SYN flood, the probability of serving any legitimate requests successfully, approaches zero. For Linux 4.19, it is notable that a small percentage ($< 10\%$) of requests is served, even for rates of up to 1 Mpps SYN flood. The profiling reveals that, in this setting, CPU utilization is the limiting factor. A clear difference between the Linux versions is the number of CPU cycles used for the hash calculation. Cookie calculation using SHA1 requires up to 15 % of the cycle budget, while SipHash (2.5 %) is more efficient.