Line-rate Botnet Detection with SmartNIC-Embedded Feature Extraction

Mario Patetta, Stefano Secci, Sami Taktak

Line Rate Botnet Detection with SmartNIC-Embedded Feature Extraction

Mario Patetta, Stefano Secci, Sami Taktak
Cnam, Paris, France. Email: first-name.last-name@cnam.fr

Abstract

Botnets pose a significant threat in network security, exacerbated by the massive adoption of vulnerable Internet-of-Things (IoT) devices. In response to that, great research effort has taken place to propose intrusion detection solutions to the botnet menace. As most techniques focus on either packet or flow granularity, port-based analysis can help detecting newly developed botnets, especially during their early propagation phase. In this paper, we introduce a line rate distributed anomaly detection system that employs NetFPGA Smart-Network Interface Cards (SmartNIC) as programmable switches. Per-port feature extraction modules are deployed directly on the data plane, enabling a centralized controller to periodically retrieve collected metrics, and feed them to a botnet detection algorithm we refine from the state of the art. We evaluate our system using real world traces spanning several months from 2016 and 2023. We show how our solutions allow keeping low the number of anomalies detected, retaining only the most relevant ones, thanks to the distributed monitoring approach that helps discriminating systemic changes from local phenomena. Furthermore, we provide an analysis of the most significant alerts, accounting for the limited ground-truth on the dataset.

Keywords: P4→NetFPGA, Software-Defined Networking, Anomaly Detection

1. Introduction

The advent of Software-Defined Networking (SDN) and programmable switching hardware has brought up the interest to delegate security tasks to the data plane. Consequently, multiple solutions have been proposed in this domain, focusing on knowledge-based [1, 2, 3, 4] and anomaly-detection-based techniques [5, 6, 7]. SmartNIC devices can offer several advantages by enabling packet processing at line rate. These benefits encompass a decrease in network and memory resource usage, as there is no need for collecting packets for offline analysis. Additionally, they facilitate a quicker response to abrupt malicious activities.

However, these works show limited effort into detecting spreading distributed attacks as botnets, as well as zero-day vulnerability exploits. Besides, they often revolve around per-flow analyses, which is intrinsically tied to very small time scales. As argued in [8], in order to catch spreading botnets, an Intrusion Detection System (IDS) should (i) adopt a port-based analysis, as it is identified as an effective way to qualify scans perpetrated during the spreading phase; (ii) profile port usage on the long-term to detect major changes in traffic patterns; (iii) have a wide view over the network to distinguish systemic anomalies from local traffic fluctuations.

Leveraging on this, we propose a system where multiple programmable switches, implemented through NetFPGA (Network Field Programmable Gate Arrays) boards, are employed to extract per-port features at line rate. These features are periodically transmitted to an external controller running a refined version of a botnet detection state of the art algorithm [8], based on statistical learning, to profile port usage over the long term. This approach enables real-time traffic analysis without the necessity to sample packets for offline processing. It conserves both bandwidth and precious server computing resources.

In particular, we design P4→NetFPGA switches integrating port-based metric extraction logic, extending the open-source project with multiple external functions ('externs'), i.e., P4's abstraction to access the functionalities of the underlying hardware. Namely, our contributions to the state of the art include:

(i) the design of a compact 16/32-bit-input and 4-kbit-bucket extern module for line rate TCP/IP header field value cardinality computation based on HyperLogLog [9] data sketch;

(ii) a lightweight approach for embedding the Welford

algorithm [10, 11] in FPGA for line rate statistical features computation, trading accuracy with lower computational complexity, leveraging on the forgiving nature of the use case, where the aim is to catch overarching pattern shifts rather than taking exact measurements;

(iii) a Southbound Interface (SBI) subsystem to collect metrics at the controller level and to enforce routing rules, including attack mitigation instructions.

We open-source the code, including an ad-hoc python controller used for the detection module, at [12]. In the analysis applied to real IP traffic traces from 2016 [13], we show how the system can detect attacks that were detected by the MAWILab algorithm [14], as well as some zero-day unknown anomalies. Moreover, we provide a detailed analysis, accounting for the limited ground-truth on the attacks, describing the evolution of features, as well as the similarities and differences in the detection performance with respect to the offline approach.

The remainder of the paper is organized as follows. Section 2 surveys the related work and describes our positioning with respect to the state of the art. Section 3 provides an overview of the proposed P4 match-action pipeline. Section 4 describes the traffic monitoring logic, while Section 5 describes the forwarding logic embedded onto the FPGA, as well as the SBI used in the proposed SDN environment. In Section 6 we present the results of the experimental evaluation using real-world traffic traces, comparing the anomaly detection performance of the proposed system with the reference state-of-the-art work and proposing an interpretation of the anomalies spotted. Finally, Section 7 concludes the paper.

2. Related Work

In this section we discuss different intrusion detection approaches, with a focus on solutions deployed in programmable hardware. Finally, we position our contribution with respect to the related work.

2.1. Intrusion Detection

Several works appear in literature to address intrusion and botnet detection [15, 16, 17]. Intrusion detection techniques can be classified as (i) knowledge-based and (ii) knowledge-agnostic approaches.

Knowledge-based solutions such as Snort [18] first analyse packets with DPI techniques, then detect malicious activity based on known patterns and signatures, which may be discovered through the use of honeypot analysis. These solutions are seldom effective in detecting zero-day threats and botnets that spread through social engineering or unknown vulnerabilities.

Knowledge-agnostic techniques aim at catching attacks regardless of them being known or unknown. The idea is to model the traffic through statistical or ML techniques and identify anomalies as significant deviations from the model. For this reason, they are hereon referred as anomaly detection techniques.

For example, HIDE [19] is an hybrid IDS based on statistical pre-processing and ML classification. BotSniffer [20] focuses on spotting spatial-temporal correlation in network traffic to identify Command-and-Control (C&C) servers and infected hosts. BotMark[21] uses k-means clustering on traffic features to determine the similarity among flows, performs a stability-based analysis to establish whether a cluster is generated by bots or not and finally considers communication graph features to identify C&C servers.

MAWILab [14] algorithm uses a graph-based method to combine the result of four detectors working at different granularities to label the MAWI Archive traffic with one out of four anomaly levels. ANTE [22] is an IDS able to automatically select which ML pipeline is best suited to detect botnets and use it to anticipate attacks. Some works [23, 24, 25] use Convolutional Neural Networks by mapping basic flow metrics to gray-scale images and extract their convolutional version to train classification models.

2.2. Hardware IDS

In the context of knowledge-based detection, several works propose hardware designs that offload Regular Expression (RE) matching (i.e., the main operation involved in DPI) to FPGA boards. Snort Offloader [1] and Pigasus [2] are two hybrid architectures where a hardware pre-filter is used to coarsely identify benign traffic and make sure that only suspicious packets are transmitted to the CPU for thorough software analysis. Authors of [3] use Non-deterministic Finite Automata approximation techniques to implement a multi-staged RE matching engine suitable for traffic beyond 100 Gbps entirely on the FPGA. Authors of [4] showcase an application of the P4→NetFPGA framework tailored to 5G multi-tenant scenarios, where TCAMS are used to implement a firewall.

Efforts in research have also targeted offloading anomaly detection tasks to the data plane. Authors of [5] propose a DDoS detection system based on Shannon entropy estimation, where several traffic metrics are extracted on a NetFPGA Sume, transmitted to the host computer and fed to a Long Short-Term Memory

Table 1: Prior work on Intrusion Detection

Reference	Type	Technique	Granularity	Hardware Offload
Snort [18]	Knowledge-based	RE matching, Rule-based detection	Hybrid	No
HIDE [19]	Anomaly detection	Statistical user profiling + ML model	Host, Network	No
BotSniffer [20]	Anomaly detection	Activity/message response detection + spatial-temporal correlation analysis	Flow, Host	No
BotMark [21]	Anomaly detection	Flow-similarity + packet-length-stability analyses, graph-based C&C detection	Flow	No
MAWILab [14]	Anomaly detection	Aggregation of any kind of anomaly detector	Hybrid	No
ANTE [22]	Anomaly detection	Autonomous ML	Flow	No
Snort Offloader [1]	Knowledge-based	RE matching	Packet	Partial
Pigasus [2]	Knowledge-based	RE matching	Packet	Partial
[3]	Knowledge-based	RE matching	Packet	Yes
[4]	Knowledge-based	Stateless Firewall	Packet	Yes
[5, 6]	Anomaly detection	Shannon entropy estimation + LSTM model	Whole traffic	Yes
FlowLens [7]	Anomaly detection	Packet length and inter packet timing analysis + ML model	Flow	Yes
DPX [26]	Hybrid	Pattern matching, bandwidth change detection and port scanning detection	Hybrid	Yes
Split-and-Merge [8]	Anomaly detection	Statistical port profiling + Modified Z-score based anomaly detection	Port	No
This contribution	Anomaly detection	Statistical port profiling + Modified Z-score based anomaly detection	Port	Yes

(LSTM) network trained to detect DDoS attacks. The same authors propose two architectures [6] where the ML-aided DDoS detection is integrated on the NetFPGA as well. FlowLens [7] leverages a novel data structure called Flow Marker Accumulator to collect per-flow packet length or inter packet timing distributions within Tofino switches. These markers are then used by a software ML algorithm running on the switch CPU to classify flows.

DPX [26] is an OpenFlow compatible data plane architecture, developed for the NetFPGA Sume board, that integrates simple security function such as payload pattern matching, detection of sudden changes in bandwidth and vertical/horizontal port scanning.

The related work is summarized in Table 1. Each contribution is characterized by: (i) the type (knowledge-based or anomaly detection); (ii) the technique used to perform intrusion detection; (iii) the granularity, i.e., the level of traffic aggregation used in the approach; (iv) whether the solution performs per-packet operations directly on the hardware - at line rate - or not.

2.3. Port Based Detection

Hardware-based network security solutions typically focus either on DPI or anomaly detection , confining themselves to packet-level or flow-level granularity. Nevertheless, they are intrinsically unable to detect overarching changes in port behavior associated with botnet spreading and novel vulnerability exploits.

Split-and-Merge [8] technique suggests an interesting approach, leveraging on a Collaborative IDS (CIDS) scheme in conjunction with a port-based unsupervised anomaly detection algorithm to detect botnets during

Table 2: Split-and-Merge Features.

Feature	Usage
Number of received packets (pktCnt)	Observe if a port is suddenly massively used
Percentage of unique src IP addresses (srcIp%)	A rise could signify a potential botnet attack, whereas a decline might suggest a sudden surge in traffic from a restricted number of hosts.
Percentage of unique dst IP addresses (dstIp%)	An increase could indicate a significant number of victims, such as in the case of a scan attack.
Percentage of unique src ports (srcPort%)	A diminution may be associated with port spoofing
Mean packet size (meanSize)	A sudden change may suggest the inception of any large scale activity
Standard deviation of packet size (stdSize)	An increase can detect the inception of an attack, while a reduction reflects a lower variety in packet size, which may be due to legitimate processes being killed on newly infected machines.
Percentage of SYN packets (Syn%)	Detect SYN scan attacks

their spreading phase, before potential attacks take place.

To accomplish this, several detection units are deployed over the network with the purpose of modeling the long-term usage of TCP ports. The port usage is represented by the features shown in Table 2, which are periodically sampled and represented as time series. Note that a port is only considered if it is interested by at least N_{min} packets within a time slot. Anomalies are detected in two steps.

First, local anomalies are identified with a change-detection algorithm based on the modified Z-score:

$$M^p_{i,W+1} = \frac{0.6745 \cdot (x^p_{i,W+1} - \tilde{f}^p_{i,W})}{median(|x^p_{i,W+1} - \tilde{f}^p_{i,W}|)} \quad (1)$$

Where $x^p_{i,W+1}$ is the data point related to the latest observation for feature i on port p, $\tilde{f}^p_{i,W}$ is the median of the last W values, and the denominator is the median absolute deviation (MAD). Using the modified Z-score, a new data point is considered anomalous if M exceeds a given threshold T_i, which is set to 3.5 as recommended in [27]. In addition, an anomaly is also identified when the *pktCnt* feature becomes greater than a given threshold N_{min} for the first time in at least W observations. This allows to detect emerging ports for which there may not be sufficient data to generate the modified Z-score, given their prior inactivity.

Local anomaly reports are then aggregated, and only those found in at least k vantage points are retained. This helps selecting the most relevant alerts, as random and local events, which are most likely not related to scanning and botnet activity, are filtered out. Finally, the Anomaly Score (AS) is computed as the total number of anomalies generated for one port, providing a compact metric to identify ports that are potentially under attack.

2.4. Our Contribution

Authors of [8] focus their efforts on designing the anomaly detection algorithm, leaving its line rate implementation for future work. In this paper, we fill that gap, deploying feature extraction logic on top of programmable switching hardware.

In a real-world scenario, implementing a Split-and-Merge-like system for anomaly detection requires either relaying the entire traffic sampled at each vantage point towards one or more dedicated servers for offline analysis, or offloading at least part of the analysis to the data plane. As the first option would cause a very inefficient bandwidth utilization for the transmission of collected packets, we focus on embedding per-port metric extraction into the data plane forwarding system.

In order to obtain the features shown in Table 2, we need to collect and store, for each packet: (i) its source and destination IP addresses (32-bit values), (ii) its source port (16-bit value), (iii) its size (16-bit value). Additionally, we need to count the number of SYN packets and the total amount of packets. Ignoring the impact of the counters, this sums up to 64-bits per packet, which can hardly scale as the throughput increases, even to monitor a single port.

For example, if we consider a 1 Gbps throughput with an average packet size of 500 bytes, and we imagine to monitor a port responsible for 0.5% of the traffic over a period of 15 minutes (as in [8]), storing 64-bits per packet header would require 9 MiB of memory. On the other hand, if the throughput is 40 Gbps (i.e., the maximum throughput of a NetFPGA switch) and the port under analysis is responsible for 2% of the traffic (which is not uncommon for a massively used port, especially

if under attack), we would need approximately 1.5 GiB of memory. Note that this is not accounting for the fact that a port being scanned would probably have a lower average packet size, resulting in more packets per second.

We work around the memory bottleneck through the use of data sketches, namely, HyperLogLog (HLL) [9] and Welford's Algorithm [10, 11]. HLL is the de facto standard algorithm for cardinality estimation. It is used, for example, in BigQuery, Google's serverless data warehouse system [28], as well as in several academic works [29, 30]. As HLL notably has poor accuracy at low cardinalities, we also employ the LinearCounting [31] algorithm to compensate for that. Welford's Algorithm is a one-pass method for computing the mean and standard deviation used in many data analysis applications [32, 33, 34].

In summary, our goal is to realize an SDN system, depicted in Figure 1, where programmable switching hardware is employed to extract port-level features from multiple vantage points. Collected features are periodically sent to the centralized SDN controller to perform anomaly detection, allowing mitigation policies to be actuated, in the form of new flow rules. Specifically, we use NetFPGA Sume boards as programmable switches, employing the P4→NetFPGA [35, 36] framework - and, in particular, the Simple Sume Switch P4 architecture it provides - to capitalize on the convenience of P4 language for extracting packet headers and the flexibility of FPGA fabric for implementing custom logic. Consequently, we designed Verilog extern functions implementing the aforementioned sketches to approximate per-port statistical features required for the anomaly detection.

Additionally, we have designed an ad-hoc SBI protocol to transfer collected features to the centralized controller. Here, anomalies related to each vantage point are derived, and consequently aggregated into network-level alerts. Finally, we designed a remotely configurable routing engine for the P4→NetFPGA architecture. Thanks to this, the controller can use the SBI to insert flow rules in the routing tables, including potential mitigation instruction. For example, traffic on a suspicious port or host can be blocked or mirrored towards a Deep Packet Inspection server for offline analysis.

3. Programmable Switch Pipeline

Figure 3 shows the proposed P4 pipeline, based on the Simple Sume Switch architecture [36]. The P4 Match-Action Pipeline is composed of two modules:

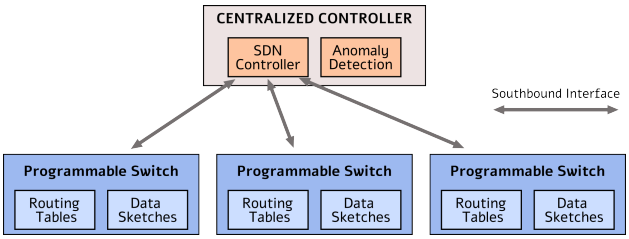


Figure 1: SDN System Overview

the Routing Engine and the Per-port Metrics Extraction Module.

In the Metrics Extraction Module, packet data obtained by the parser is fed to the data sketch extern function modules. Specifically, HLL modules are used to estimate the cardinality of IP source and destination addresses, as well as TCP source ports, while the Welford module includes the logic to iteratively update the mean and standard deviation of the packet size, as well as two counters to track the number of SYN packets and the total packet count. Finally, the TCP Port Register keeps track of the port currently under evaluation, and it can be configured by the controller with an SBI Set TCP Port message.

The Routing Engine handles the forwarding decision, based on the Routing Table extern function modules, that include remotely configurable Lookup Tables (LUT) used to match packet headers to destination ports. Specifically, there is one routing table for SBI messages and four tables for routing data plane traffic according to IP src/dst addresses and TCP src/dst ports.

Two Data Flows can be identified, as shown in Figure 2. First, packets are parsed into two categories: SBI messages and TCP traffic.

For SBI messages, their Switch ID is compared to the one stored in the Switch ID register. If there is no match, it means that the message is destined to another device, and the packet must be forwarded according to the SBI table. Conversely, if there is a match, it means that the message contains an instruction intended for this switch, and it must be processed according to the Message Type, as described in Section 5. Finally, the Up-link flag is set to 1 to mark that, from now on, the SBI message is destined to the controller, and the destination port is set to be the same as the source port, so that the packet is bounced back upstream.

For TCP traffic, IP and TCP ports are first checked in the data plane forwarding tables, to determine the destination port. Then, if the TCP destination port corresponds to the one currently under analysis, packet information, i.e., [Src IP address, Dst IP address, Src TCP

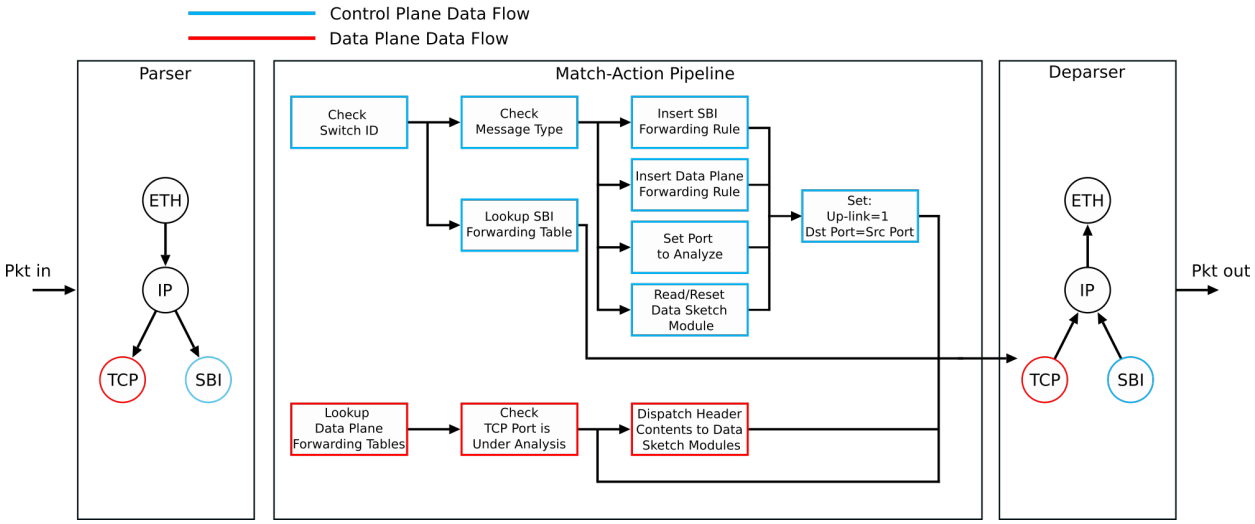


Figure 2: P4 Pipeline deployed in the Simple Sume Switch Architecture

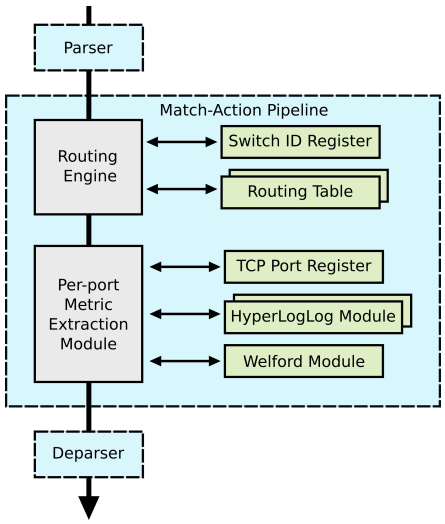


Figure 3: P4 Architecture

port, pkt size, SYN flag], is dispatched to the data sketch extern modules.

4. Data Sketch Modules

In this section, we go over the design of the metric extraction extern functions, i.e., HLL and Welford modules.

4.1. HyperLogLog *Extern Module*

HLL is a cardinality estimation algorithm based on hash functions. If the output of a hash function is uniformly distributed (i.e., the probability of each bit to be 0 or 1 is the same), then hash values with considerable consecutive leading zeroes are rare. Therefore, the cardinality of a dataset can be estimated by looking at the rarest value obtained from feeding its elements to the hash function. If ρ is the largest leading one position among the hash values extracted from a dataset, then a good approximation of its cardinality is 2^{ρ}.

To reduce the variability of the estimation, we can partition the dataset in multiple streams and maintain distinct values of ρ in m separate buckets. The cardinality is then calculated as the normalized harmonic mean of the buckets, using negative powers of two instead of reciprocals.

$$E_{HLL} := \underbrace{\frac{0.7213}{(1+1.079/m)}}_{\alpha_m} \cdot m^2 \cdot \underbrace{\left(\sum_{j=1}^{m} 2^{-\rho_j}\right)^{-1}}_{Z} \tag{2}$$

4.1.1. Low Cardinality Correction

For cardinalities below $\frac{5}{2}m$, nonlinear distortions appear in the estimation. For example, when all the buckets are initialised to 0, Equation 2 returns $\alpha_m m^2 m^{-1} = \alpha_m m \approx 0.72m$. To compensate for this, is it common to use LinearCounting [31], another cardinality estimation algorithm that is more precise for low cardinalities but less scalable. The principle behind LinearCounting is that, if we randomly throw n balls into m buckets, the expected amount of empty buckets is $m \cdot exp(-\frac{n}{m})$. Hence, if we observe V empty buckets, a good estimation of the cardinality is:

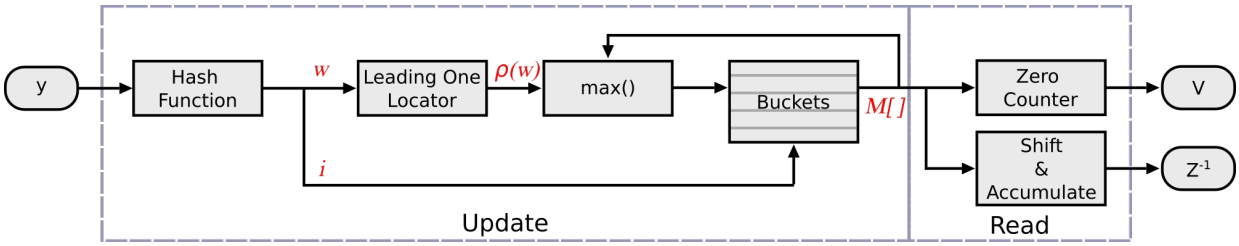


Figure 4: Block Diagram of the HyperLogLog Extern Function

$$E_{LC} \coloneqq m \cdot \log\left(\frac{m}{V}\right) \tag{3}$$

HLL can be easily extended to employ LinearCounting for low cardinalities, as we can use the same set of buckets to determine both V and Z, i.e., the first is the number of empty buckets, while the second is the normalized harmonic mean of the buckets.

4.1.2. Extern Design

Recently, an FPGA implementation of HLL for networking applications has been proposed in [37]. One of the goals of this work is to achieve very high cardinalities (i.e., over 10^8) and maximize the estimation accuracy. Accordingly, a 64-bit hash function, along with a total of 2^{16} buckets are employed. For our use case, high accuracy is of less concern, as our final objective is to identify significant shifts in traffic patterns. Also, since we partition the traffic both according to TCP destination ports and in time slots of few minutes, we do not expect to face such high cardinalities. For these reasons, we designed a lightweight version of HLL, employing a custom 23-bit hash function and 2^8 buckets. Furthermore, we embrace the principle of offloading to the data plane only the operations that are required to run at line rate, leaving to the controller those that can be executed in software. Consequently, we don't integrate the logic to solve Equations 2 and 3, but limit ourselves to compute Z^{-1} and V. In fact, Z^{-1} and V are updated for every packet on the TCP port under analysis, while Equations 2 and 3 should only be solved once per time slot. This has no impact on the amount of data to store and to transfer to the controller, but allows to save FPGA resources.

Figure 4 shows a functional block diagram of the HLL extern module. The operations performed by the module are dictated by the commands received from the P4 pipeline: *hllUpdate*, *hllRead* or *hllReset*. While *hllUpdate* is applied when data plane packets match the TCP destination port under analysis, *hllRead* and *hllReset* are triggered by specific Metric Query control plane messages.

In the following, we will describe the behavior of the HLL extern module under the three possible operational modes.

Reset. Upon receiving the command *hllReset*, the module enters a loop where the content of each bucket is systematically set to 0.

Update. When *hllUpdate* is issued, the sketch data structure is updated based on the input y, i.e., the source or destination IP address, or the source TCP port. Initially, y is fed to a hashing module. To obtain a uniformly distributed 23-bit string output, we designed a custom multiplication-shift-XOR based function.

The purpose of using multiplication is to diffuse bits leftwards, whereas by shifting we redistribute them back rightwards, effectively shuffling them to obtain uniformly distributed hash values. To realize the hash function, the input is split and padded into two 24-bit words, each multiplied by distinct 18-bit odd constant seeds, right shifted by a constant amount and XORred into a single 23-bit word. The size of the multiplication operands are chosen according to the capabilities of the 24x18 multiplier present in the DSP48E1 slices on the FPGA, as specified in [38]. We fix the output length to 23 bits, as we need an 8-bit index i to address the 256 buckets, and a 15-bit word w since each 4-bit bucket can contain a leading one position up to $2^4 - 1$.

To determine $\rho(w)$, i.e., the position of the leading 1 in w, we designed the Leading One Locator module. Here, a cascade of right-shifters and bitwise-ORs iteratively converts to 1 all the bits after the leftmost one. Then, the Hamming weight HW is computed by summing the individual bits of the resulting string. Finally, $\rho(w) = 15 - HW + 1$, as the leading one can be seen as the number of leading zeros (i.e., the length of w minus HW) plus one.

For example, if w is $'000101010000111'$, the shift-OR cascade yields $'000111111111111'$, the Hamming Weight is 12 and finally $\rho(w) = 15 - 12 + 1 = 4$ (which is indeed the position of the leading one in w).

Algorithm 1: Welford's Algorithm

input : A set X of N values $x_i \in \{x_1, ..., x_N\}$
output: The number N, the average avg and the variance var of the elements in X

Function *Update*(N, avg, M_2, x)**:**
  $N \leftarrow N + 1$;
  $\delta_1 \leftarrow x - avg$;
  $avg \leftarrow avg + \delta_1/N$;
  $\delta_2 \leftarrow x - avg$;
  $M_2 \leftarrow M_2 + \delta_1 \cdot \delta_2$;
return N, avg, M_2
Function *Welford*(X)**:**
  for $x_i \in \{x_1, ..., x_N\}$ **do**
    $(N, avg, M_2) \leftarrow$*Update*(N, avg, M_2, x_i);
  end
  $var \leftarrow M_2/(N-1)$
return N, avg, var

Lastly, $\rho(w)$ is compared with $M[i]$, the value currently stored in the i-th bucket, and the max value is retained.

Read. When *hllRead* is issued, the buckets are systematically read, in order to compute Z^{-1} and V, and transmit them to the P4 pipeline to craft the Metric Query response message. Accordingly, values of M are fed to a right shifter to obtain the related negative powers of two, and then summed up to obtain Z^{-1}. Simultaneously, a counter is employed to determine the number of empty buckets V.

Upon receiving the Metric Query response, the controller will translate Z^{-1} to an actual cardinality estimate by using Equation 2, and eventually use V paired with Equation 3 in case the first estimation is lower than $\frac{5}{2}m$.

As already mentioned, each HLL module employs 256 4-bit buckets, with a memory footprint of just 4 kbits of FPGA BRAM resources, and it is able to estimate cardinalities up to $\alpha_{256} \cdot 256^2/\left(256 \cdot 2^{-15}\right) \approx 6 \times 10^6$. Moreover, the read operation consists of a fixed and compact amount of shifts and sums compared to a non-sketch approach, consisting in storing data for all the packets and periodically counting the unique elements. This option not only requires much more memory, but also concentrates an amount of operations proportional to the volume of packets at the end of each time slot, dramatically increasing the duration of the read operation, during which the module is unavailable.

4.2. Welford's Algorithm Extern Module

Welford's algorithm is a one-pass method to estimate mean and variance with great accuracy, without needing to store previously observed data.

As shown in Algorithm 1, for each new value x_i, three variables are iteratively updated: the total number of observations N, the average estimation avg and the squared sum of differences from the mean M_2. At each iteration, N is incremented by 1; avg is incremented by the ratio between δ_1 (i.e., the difference between the new value and the current mean) and N; while M_2 is incremented by the product between δ_1 and δ_2 (i.e., the difference between the new value and the updated mean). At any given moment, the three values can be used to determine the mean and variance of the data observed so far. While avg is a direct estimation of the mean, the estimated variance can be obtained by dividing M_2 by $N - 1$.

4.2.1. Extern Design

Recently, some works have proposed promising ways to implement Welford's Algorithm in FPGA, tackling the expensive hardware deployment of the division operation. In [39], the division is simplified by using a multiply-shift approach [40] and a LUT to store all possible division values. This is feasible as the range of potential divisors is limited to integer values between 1 and 255. Authors of [41] designed an efficient hardware architecture for integer division, that rounds the result to the closest power of two.

While the multiply-shift approach is not suitable to our use case, as it would require a LUT as big as the amount of possible values of the divisor (i.e., 2^{24}), the design proposed in [41] could fit our needs. Nevertheless, we opt for a custom, more lightweight design, coarsely approximating the division with a right shift. This choice is driven by two factors: (i) we prioritize resource usage over accuracy, as we aim to catch long-term changes and not fine grained values; (ii) we can reuse the Verilog code of the leading one locator employed in the HLL extern to find the number of bits to right shift.

As shown in Figure 5, the Welford Extern module comprises four registers. Two registers are used to count the number of packets N and the number of SYN packets, the remaining two registers are used to keep track of the variables avg and M_2, required to estimate the mean and standard deviation of the packet size. The conversion from M_2 to std is left to the controller, as we only need to perform it once per time slot.

Similarly to the HLL extern, three operations are defined: $welfUpdate$, $welfRead$ and $welfReset$.

welfUpdate is applied to every data plane traffic matching the TCP destination port under analysis, whereas *welfRead* and *welfReset* are triggered by specific control plane messages.

Reset. When *welfReset* is issued, the module simply sets to 0 the four registers.

Update. Along with the *welfUpdate* command, the P4 pipeline provides the inputs *Pkt-len*, representing the size of the current packet, and *SYN-flag*, which is set to 1 if the packet's SYN flag is raised or 0 otherwise. The logic to update N and the SYN registers consists of two counters. The first counter is triggered by each *welfUpdate*, while the second is activated if the *SYN-flag* is up. To update *avg*, its current value is incremented by δ_1/N, where δ_1 is the difference between *avg* and x. The division by N is approximated by a right shift of k bits, where 2^k is the power of two closest to N. Similarly to δ_1, δ_2 is obtained as the difference between x and the updated value of *avg*. Subsequently, the two deltas are multiplied, and the results are accumulated in the M_2 register.

Read. Upon receiving a *welfRead* command, the module transmits the content of its registers to the P4 pipeline for filling the related fields in the *Metric Query* message before bouncing it back to the controller.

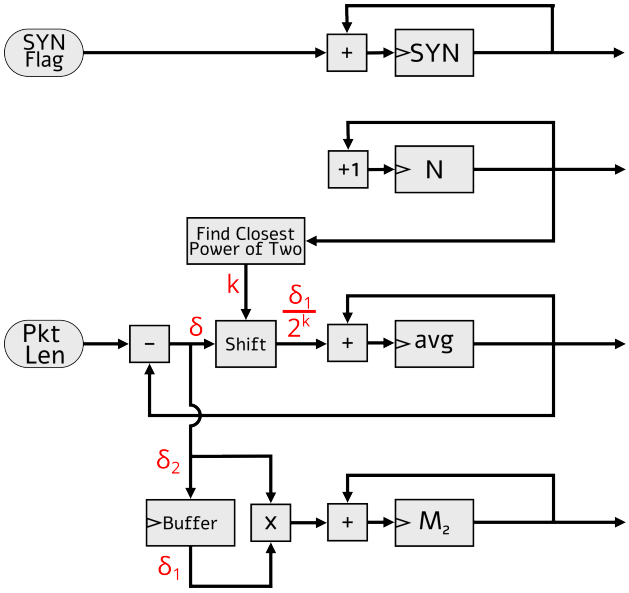


Figure 5: Block Diagram of the Welford Extern Function, including a SYN packets counter

4.2.2. Division Approximation

Division is not supported by DSP48E1 slices [38] available on the board and nontrivial to implement in hardware, often resulting in designs that take a lot of space and cause great latency. As out goal is to catch anomalies by detecting major changes in the network traffic, we care little about fine graded measurements. Therefore, we opt for a coarse yet lightweight approach by substituting the division with the right shift operation. Right shifting a number by k-bits is equivalent to dividing it by 2^k. Hence, if k is chosen such that 2^k is the power of two closest to N, the relative error is

$$\begin{aligned} Err(N) &= 1 - \frac{\left(\frac{\delta_1}{N}\right)_{appr}}{\left(\frac{\delta_1}{N}\right)_{exact}} = 1 - \frac{\frac{\delta_1}{2^k}}{\frac{\delta_1}{N}} \\ &= 1 - \frac{N}{2^k} = 1 - \frac{2^k + r}{2^k} = \frac{r}{2^k} \end{aligned} \tag{4}$$

Where r is the difference between N and its closest power of two, such that $N = 2^k + r$. Figure 6 shows the approximation errors for different values of N. The error is roughly bounded between -30% and 40%. As N increases by 1 at each iteration, we will experience all the possible values of $Err(N)$. Hence, negative errors will partially compensate for positive ones as the results of the division are accumulated to estimate *avg*. Moreover, the cumulative error needs to be small enough only for the first iterations, as the ratio δ_1/N naturally decreases as N increases (being δ_1 limited), resulting in that even large relative errors do not have a strong impact on the final estimation.

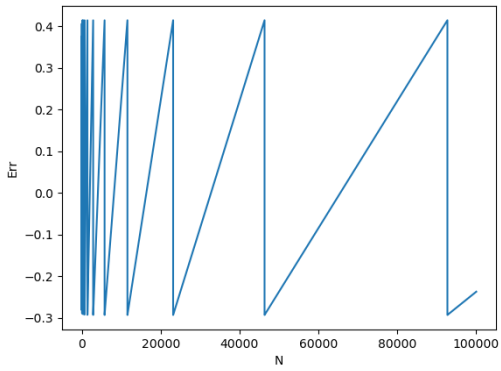


Figure 6: Relative Error Using Right Shift Instead of Division

To obtain k from N, we first convert to 1 all its bits after the leading one as in the HLL module, obtaining the string s. We then get the next power of two 2^l as $s + 1$, and the previous one 2^{l-1} is obtained by right

shifting it by one. To find out which one is closer to N we just compare the differences $2^l - N$ and $N - 2^{l-1}$. If the former is smaller, than k is equal to the Hamming Weight of s, otherwise it is that value minus one.

For example, if $N = 20$, that in binary corresponds to $'10100'$ (note that we omit the first 19 bits set to zero for the sake of simplicity), then $s =' 11111'$. Hence, the next power of two is $2^l = s + 1 =' 100000' = 32$, and the previous one is $2^{l-1} = 16$. As the difference between N and 2^{l-1} is lower than the difference between 2^l and N, $2^{l-1} = 16$ is the closest power of two, and k is equal to the Hamming Weight of s minus one, i.e., $k = 4$.

5. Southbound Interface and Routing Engine

Two requirements of the proposed IDS are the capability to retrieve metrics computed by multiple Switches to detect anomalies, and the capability to enforce routing rules tailored to the observed threats (e.g., blocking an anomalous port, forwarding traffic towards a dedicated server for offline analysis or blocking specific IP addresses). Both these actions require the control plane to be able to communicate with the data plane.

As most popular SBI solutions [42, 43] are not natively supported by the P4→NetFPGA framework, we achieve the remote controllability of the Sume Switch through an SBI derived from the one proposed in [44], adapted to provide the specific messages required in our IDS. In the remaining of this section we will first present the set of control messages composing the SBI and then we describe the Routing Engine module.

5.1. SBI Messages

In the proposed SBI, every message is carried by an IP packet (protocol ID 144) and is composed by a common shell, plus the content. The SBI shell carries five fields:

(i) the Controller ID field is introduced for scenarios where multiple controllers are involved; (ii) the Switch ID is used to identify the target of a message, allowing switches to assess if they are the intended recipient and to decide where to forward the packet; (iii) the Up-link flag is used to distinguish SBI messages that are directed from the Control Plane to the Data Plane and vice versa; (iv) the Type field identifies the content of the message, and determines how the SBI packet will be treated by the P4 pipeline; (v) the Length corresponds to the number of bytes composing the message, including the shell.

SBI message forwarding is managed by a dedicated SBI routing table that maps [Switch ID, Up-link flag] pairs to destination ports. Whenever a Sume Switch receives a message destined to itself, it will process it accordingly and send it back to the port it came from with the Up-link flag set. This way, the controller will receive confirmation of receipt for the messages sent, along with eventual complementary information included in the message (e.g., metric query).

Figure 7 shows the SBI shell format, as well as the proposed set of control messages:

1. *Set Switch ID* messages are used to change the ID of the receiving device, saved in the Switch ID register. Each device only responds to messages destined to its Switch ID, and forward the rest according to SBI routing tables. Every switch starts with an ID equal to zero.
2. *SBI Routing* messages are used to configure new entries in the SBI routing table.
   Upon reception, the switch stores a new rule at the line specified by the LUT Address field. Rules map [Switch ID, Up-link flag] pairs, corresponding to the [Key Low, Key High] fields carried by SBI messages, to a given output port, as specified by the Port field. The higher the address a rule is stored in, the higher its priority. If the Check flag is set, no rule is added. Instead, the switch searches for the given keys in the table and eventually writes the matching port in the Port field
   before bouncing the message back to the controller. Such messages can be used to verify the correct configuration of routing tables.
3. *Destination* and *Source IP Routing* messages, similarly to *SBI Routing messages*, are used to populate the related tables. In this case, Keys correspond to IP addresses, while the Mask is used to express wildcard bits,
   i.e., bits that can have any value and still result in a match.
4. *Destination* and *Source TCP Routing* messages are used to configure the related routing tables. Keys correspond to TCP ports.
5. *Set TCP Port* messages are used to configure the destination TCP port under analysis. The current design allows to monitor one port at a time, that is configured by the controller.
6. *Metric Query* messages are used to retrieve the metrics stored in the data sketch extern modules. The Metric ID corresponds to the metric whose value is being queried. When it is 0 (*Src IP Cardinality*), 1 (*Dst IP Cardinality*) or 2 (*Src Port Cardinality*), the switch will fill Result 1 and 2 respectively with Z^-1 and V from the related HLL module before send-

ing the message back to the controller, while Result 3 and 4 are unused. When the Metric ID is set to 3 (*Counters*), Result 1 is filled with the content of the SYN counter, while Result 2,3 and 4 are respectively filled with N, avg and M, from the Welford extern. Finally, if the Reset flag is set, the related extern module is reset instead.

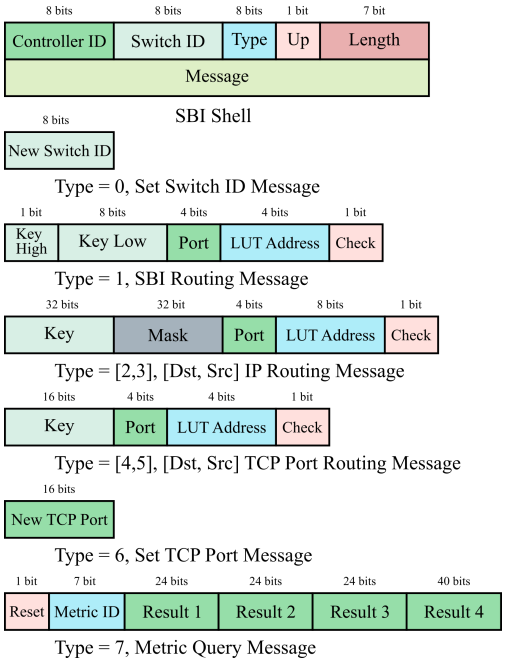


Figure 7: Southbound Interface Message Format

5.2. *Routing Engine*

The Routing Engine is responsible for the routing decision. As shown in Figure 8, it is composed of three elements: the SBI Engine, the Data Plane Traffic Engine and the Routing Decision Module.

The SBI Engine is transparent with respect to regular data plane traffic, but is triggered by control plane messages. First, it checks if the Switch ID of the incoming SBI message matches the ID of the switch itself. If the ID checks, a P4 metadata signal *id_match* is sent to the cascading modules, to enable them to process the message accordingly.

By contrast, if the ID does not check, the SBI Engine proceeds to search for the related [Switch ID, Up-link flag] pair in the SBI Routing Table, to retrieve the corresponding output port and transmit it to the Routing Decision Module.

The Data Plane Traffic Engine acts as interface between the P4 pipeline and the routing tables. For each data plane packet, its source and destination IP addresses are queried in the respective routing table and eventual matching output ports are sent to the Routing Decision Module. In case of TCP packets, source and destination ports are searched as well.

The Routing Decision Module is responsible for choosing the output port based on the information received from the routing tables. If the *id_match* signal is active (i.e., the packet is an SBI message destined to this switch), the Up-link flag is raised and the destination port is set as the source port. If *id_match* is not active and there is a match in the SBI Routing Table, its output is set as destination port. If the packet is a data plane packet, the destination port is set to the output of the highest priority data plane routing table that scores a match. The order of priority, from high to low, is: destination TCP port, source TCP port, source IP, destination IP. The idea is to use the destination IP address as default routing principle, while the other tables are used for filtering traffic and blocklisting individual hosts.

6. Evaluation

In this section, we evaluate different aspects of the proposed solution. First, we analyse the estimation

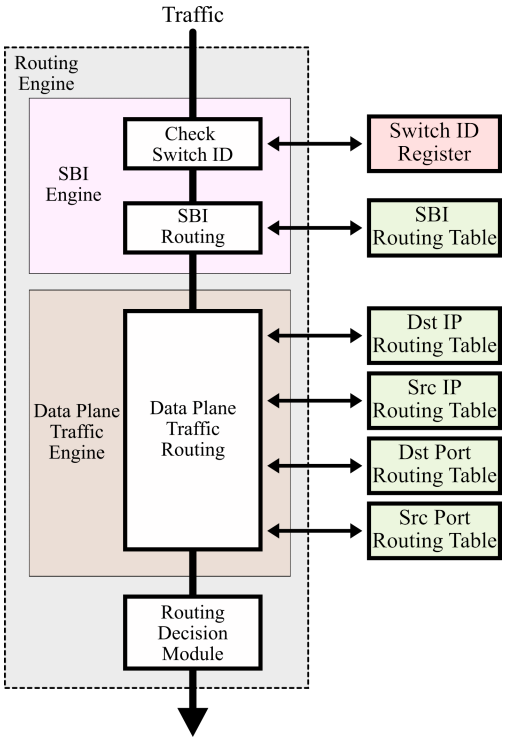


Figure 8: Routing Engine Module

performance of the designed data sketches, comparing them to the respective software implementations. Then, we showcase the effectiveness of the IDS by using it to detect anomalies in real-world traffic. Finally, we discuss scalability and resource usage of the proposed solution.

6.1. Data Sketch Estimation Error

To determine the estimation errors of the proposed HLL and Welford implementations, we crafted packet distributions with ad-hoc IP cardinalities and packet lengths. We repeated each test 50 times, and computed the related standard error of estimation.

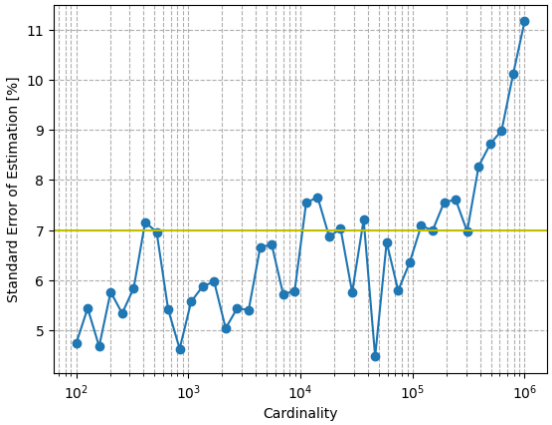
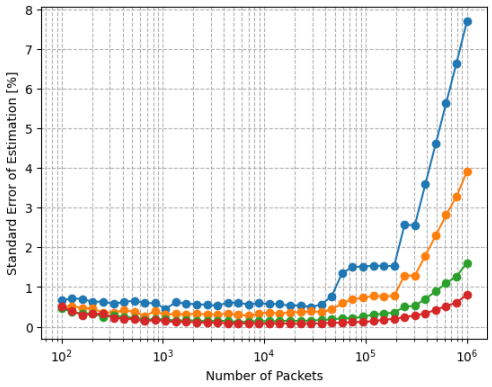


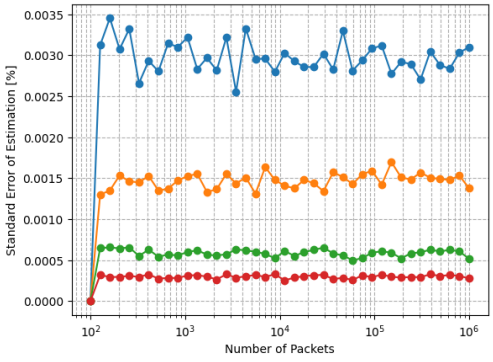
Figure 9: Standard Error of the HLL Sketch. The horizontal line represents the predicted standard error.

6.1.1. HLL Evaluation

Figure 9 shows the Standard Error of Estimation of the HLL module, for increasing cardinalities. The horizontal line represents the theoretical standard error for 256 buckets [9] (i.e., 7%). The measured error is mostly below the predicted one, except for very high cardinalities, as we approach bucket saturation and hash collisions cause new elements to be interpreted as already seen ones. According to [9], the value at which hash collisions should start affecting the estimation is 2^L, where L is the length of the hashed word. In our implementation, $L = 15$, so hashing collisions should start being relevant at cardinalities around 30k, but this effect seems to be quite moderate up until values grater than 100k. Note that, even at cardinalities close to one million, the standard error of estimation is around 11%, which is still acceptable for our use case, where the goal is to identify major changes rather then accurate measurements.



(a) Hardware Implementation



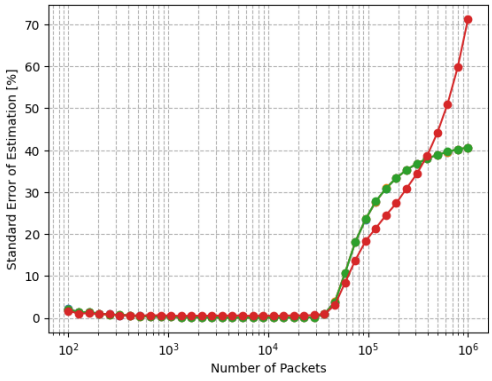
(b) Software Implementation

Figure 10: Standard Error of the Welford Sketch Estimation of the Mean. Each curve corresponds to a different value of mean: 100 (blue), 200 (yellow), 500 (green), 1000 (red).

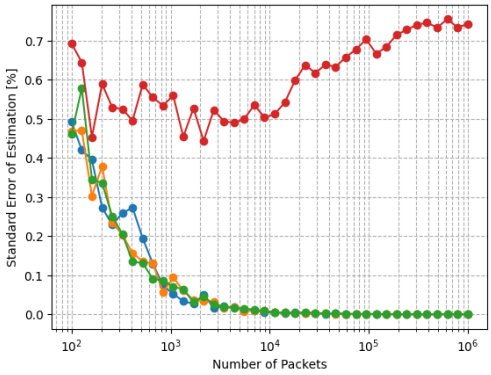
6.1.2. Welford Evaluation

As Welford's Algorithm module is used to estimate two features, i.e., packet size mean and standard deviation, we have more degrees of freedom with respect to the HLL analysis. Hence, we perform two distinct analyses: one to assess the estimation error for the mean and one for the standard deviation. As no theoretical boundaries are provided in Welford's paper [10], we compare our results with a software implementation of the algorithm.

In the mean estimation performance analysis, we repeat the test for four mean values and increasing number of packets, keeping the standard deviation equal to 20% of the mean. Figure 10 shows the Standard Error of Estimation of the mean packet size for packet distributions with increasing number of packets and mean size equal to 100, 200, 500 and 1000 bytes respectively. In both cases, hardware and software, we observe smaller errors as the real mean increases. Differently from the software version, which exhibits stable and negligible standard errors, the hardware implementation provides



(a) Hardware Implementation



(b) Software Implementation

Figure 11: Standard Error of the Welford Sketch Estimation of the Standard Deviation. Each curve corresponds to a different standard deviation: 25 (blue), 50 (yellow), 100 (green), 250 (red).

a good approximation of the mean size, with standard errors lower than 1%, when the number of packets is lower than few hundred thousands, and starts diverging afterwards. Interestingly, the standard error is more stable as the mean increases. Note that the worst case error is still lower than 8% even at 1 million packets, which is perfectly in line with our use case requirements.

Similarly, to evaluate the performance of the standard deviation estimation, we fix the mean to 500 bytes and show the results of four standard deviations (i.e., 25, 50, 100 and 250 bytes) and increasing number of packets in Figure 11. Again, the software version exhibits errors consistently below 1%. We observe that, when the standard deviation is 250 bytes (i.e., 50% of the average packet size, that we fixed at 500 bytes), the standard error of the software version increases with the number of packets, although it remains under 1%. Our design too exhibits minimal standard errors for distributions with a number of packets lower than few tens of thousands, which then rapidly degrade as the number of packet increases. The case with standard deviation equal to 250 bytes is anomalous even in the hardware case, where it diverges much faster than the others, whose curves mostly overlap.

In conclusion, we assess that the Welford Extern comes with negligible standard errors for distributions with less than few tens of thousands of packets. Afterwards, the estimation of the mean starts to slowly degrade but remains acceptable for our use case. On the other hand, the estimation of the standard deviation degrades much faster, most likely having a non-negligible impact on the capability to detect anomalies on that feature. Note that, as elaborated in Section 6.3, some correction strategies can be employed by the controller.

6.2. Anomaly Detection using Real-World Traffic

We now evaluate the proposed IDS using real-world traces. The experimental setup comprises a server connected to one of the ports of a P4→NetFPGA standalone switch. The server is responsible for (i) controlling the switch, configuring ports to analyse, querying and resetting metrics by using the SBI messages described in Section 5; (ii) replaying the traffic traces.

In the following, we briefly describe the dataset that we have used. In particular, we explain how we manage to divide each trace by the respective subnetwork, enabling us to analyse the dataset as a Collaborative IDS. Accordingly, we provide an example of the collaborative approach by showing the evolution of one of the monitored features over time for several subnetworks. Then, we run the anomaly detection over two periods spanning several months. The first one is related to 2016, in order to compare our results with the ones proposed in [8], while the second one is from 2023, to assess the effectiveness of the approach in a more recent time.

6.2.1. MAWI Dataset

The WIDE Working group maintains a repository of daily traces of a transpacific link, namely the MAWI archive [13]. Traces are collected between their network and the upstream ISP, between 14:00:00 and 14:15:00, and are anonymised so that no personal information can be extracted. Specifically, application data is removed and IP addresses are scrambled following two principles: 1) there is a one-to-one mapping between IP addresses before and after anonymisation; 2) it is prefix-preserving. This enables to retrieve the subnetworks after anonymisation. Therefore, we are able to split traces by destination IP into nine subnetworks and feed them individually to the NetFPGA Sume board to mimic a configuration with nine distinct switches.

To compare ourselves with [8], we first analyse each Thursday over a period from March 31 to December 29, 2016. For this analysis, we set the time series length W to 10, the modified Z-score threshold T_i to 3.5, and the minimum number of appearances for a local anomaly to be considered as a global one k to 2. Consequently, the first ten weeks are needed to form the initial time series and the detection begins on June 9. We repeat the analysis considering two values for the parameter N_{min}, i.e., the minimum amount of packets for a port to be considered used in a given time slot.

Additionally, we perform a similar analysis over a more recent period from January 12 to July 27, 2023. Using the same parameters, the first ten weeks are also needed to populate the time series and the detection begins on March 23.

6.2.2. Per-Feature Anomaly Detection

Figure 12 displays the evolution of the modified Z-score for the feature *srcIp%* on port 3389 in 2016. Only five subnets appear, since not enough traffic (less than $N_{min} = 100$ packets) was measured for the remaining four on that port. We can see that the threshold $-T_i$ is crossed on Jul. 28, Aug. 11 and 25, Sept. 28 and Oct. 20, but only in the last two occurrences this happens in at least two subnets. Hence, the threshold crossings of July and August are not accounted for when deriving the AS of port 3389, as localized events are most likely not related to scanning and botnet activity.

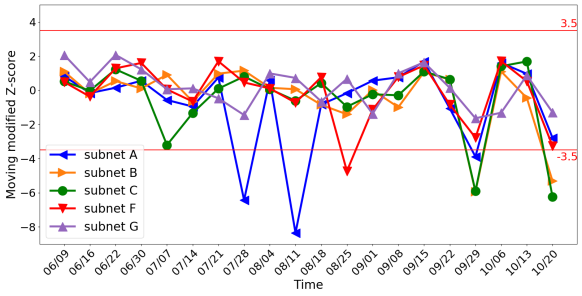


Figure 12: Evolution of the Modified Z-score for the Feature *srcIp%* on port 3389 in 2016.

6.2.3. Detection in 2016

We now analyse the metrics collected for ports that were interested by at least 2000 packets per trace for at least 3 traces in the observation period. We compare the anomalies detected in two cases: when $N_{min} = 20$ and $N_{min} = 100$. Note that we are able to analyse both cases without needing to run the traces twice because the metric collection performed by the NetFPGA programmable switch is agnostic with respect to the anomaly detection algorithm. We can tweak Split-and-Merge parameters and could even integrate other anomaly detection algorithms based on the same metrics.

Figure 13 shows the AS occurrences by day for the two cases. The numbers noted in the squares indicate the number of ports with a given AS on that day. On average, the number of daily alerts with AS grater than 15 is 0.47, when $N_{min} = 100$, and 1 when $N_{min} = 20$. This means that the average amount of alerts occurring per day is minimal, which is convenient for network administrators.

Table 3 includes all the anomalies with AS greater than 15, specifying the scores obtained in the two cases $N_{min} = 20$ and $N_{min} = 100$. The table also includes the eventual score reported in the original Split-and-Merge works for the same anomaly and the list of anomalies detected by MAWILab [14] on the same port in that day, based on the taxonomy proposed in [45]. Finally, the percentage of TCP packets related to that port is included to show that we are able to spot anomalies even on lightly used protocols.

The notion of false positive in zero-day attacks is not standard, as these attacks are unknown and therefore impossible to classify. Concurrently, as detected anomalies reflect sudden changes in the overall traffic pattern, which may or may not be due to malicious activity, there is no solid ground-truth for assigning a label to each event. Nevertheless, we investigate major anomalies, trying to match the profile of the anomalous features with known vulnerabilities reported on the related port. In the following, we present some examples.

1. *June 30 - port 6379.* This anomaly corresponds to the Redis scan attack also spotted in [8, 46]. Redis is an in-memory data store for which several vulnerabilities were discovered few days before this event (CVE-2016-8339, CVE-2016-10517). The anomaly pattern consists in a drop in *srcIp%* and a spike in *dstIp%* and *srcPort%*. Hence, we can infer that it is a scan attack to find vulnerable devices.
2. *July 7 - port 445.* We have a spike in *dstIp%* and *stdSize*, along with a reduction in *srcPort%*, suggesting a strong presence of crafted packets and botnet activity. Port 445 is used by the Server Message Block (SMB) protocol, a file sharing protocol developed by Microsoft, and known to be vulnerable to plenty of trojans and worms. MAWILab reports a scan attack on this port in the same day.
3. *July 7 - port 995.* For $N_{min} = 100$ we have a spike in *dstIp%*, *srcPort%* and *Syn%* in three subnets, it is

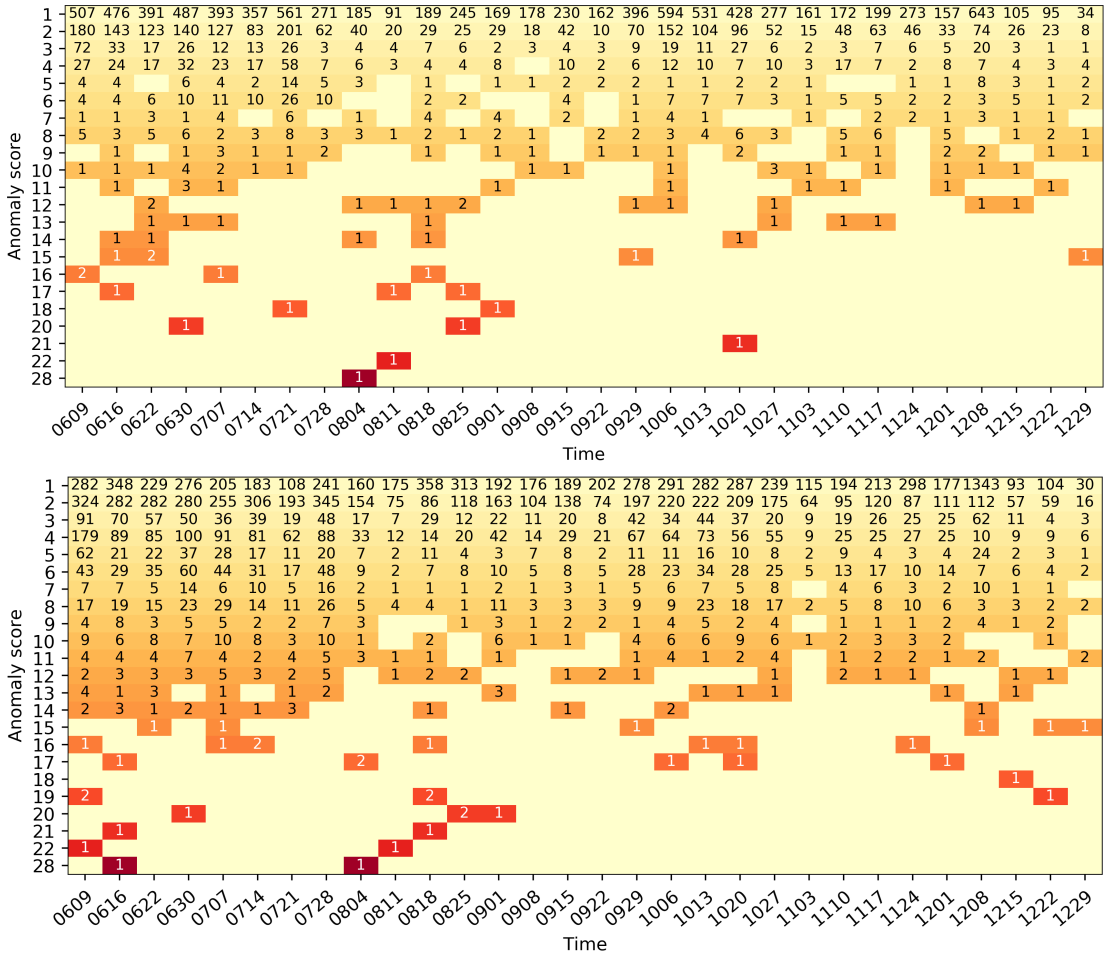


Figure 13: Anomaly Scores for the 2016 period. N_{min} = 100 (up) N_{min} = 20 (down)

an emerging port (i.e., the traffic was lower than N_{min} before this day) in another subnet and is practically unused in the others. The port is used by the POP3 mail over SSL service, but is also used in the Cyclops Blink Botnet developed by the Russian group Sandworm. The anomaly pattern for N_{min} = 100 may indicate a stealthy scan, even though the activity of this botnet was only reported starting from 2019.

4. *August 4 - port 23*. This anomaly corresponds to the Mirai [47] scan also spotted in [8, 46]. Infected hosts scan for vulnerable machines by sending random TCP SYN packets on Telnet ports 23 and 2323. It is characterised by a surge in *srcIp%*, *srcPort%*, *stdSize* and *Syn%*, along with a drop in *meanSize*. Interestingly, no spike in *dstIp%* is recorded, as Telnet ports were already massively scanned even before the outbreak of Mirai. The same scan is also detected on August 11 and 25.
5. *August 18 - port 3128*. This port is used by a web proxy service called Squid, for which several vulnerabilities leading to privacy violations (CVE-2016-10003), cache poisoning (CVE-2016-4553), arbitrary code execution (CVE-2016-4054), and denial of service (CVE-2016-4052) were discovered in the same year. As the anomaly pattern shows a surge in *srcIp%*, along with a reduction in *dstIp%* and *Syn%*, it could be an intrusion attempt perpetrated by multiple machines towards Squid servers located in the MAWI network. The same anomaly is detected on September 1.
6. *August 18 - port 52869*. The anomaly pattern shows a surge in *dstIp%*, *srcPort%* and *Syn%*, which may be in line with a stealthy scan to detect Realtek Universal Plug and Play (UPnP) devices accessible through UPnP over SOAP on port 52869. A vulnerability on this service (CVE-2014-8361) allows remote attackers to execute arbitrary code.
This is most likely related to the Satori botnet, a Mirai variant operating on ports 37215 and 52869. Interestingly, the propagation stage of this botnet

Table 3: Detected Anomalies with $AS > 15$.

Date	Dst Port	AS ($N_{min} = 20$)	AS ($N_{min} = 100$)	AS (Original)	MAWILab Report	TCP traffic %
Jun 6	81	19	16		scan	0.1%
Jun 6	1234	22				< 0.001%
Jun 6	2222	19	16			< 0.1%
Jun 6	54126	16				< 0.001%
Jun 16	81	28	17			< 0.1%
Jun 16	1234	21				< 0.001%
Jun 16	2222	17			multi-point	< 0.1%
Jun 30	6379	20	20	20	missing report	0.1%
Jul 7	445	16			scan	0.7%
Jul 7	995	15	16			< 0.01%
Jul 14	49562	16				< 0.0001%
Jul 14	52160	16				< 0.0001%
Aug 4	23	28	28	26	scan, multi-point	4.3%
Aug 4	49320	17				0.1%
Aug 4	50000	17				0.01%
Aug 11	23	22	22		scan, multi-point	4.5%
Aug 11	587		17			0.02%
Aug 18	3128	21	16			0.03%
Aug 18	8118	16				0.02%
Aug 18	50304	19				< 0.01%
Aug 18	52869	19				< 0.01%
Aug 25	21	20	17			0.2%
Aug 25	23	20	20		scan, multi-point, alpha-flow	4.5%
Sep 1	3128	20	18		scan	0.03%
Sep 29	3389	15	15	16	missing report	0.1%
Oct 6	1080	17				< 0.001%
Oct 13	50288	16				< 0.001%
Oct 20	3389	16	21		missing report	0.18%
Oct 20	18245	17			missing report	< 0.001%
Nov 24	1723	16			missing report	< 0.001%
Dec 1	5555	17			missing report	0.35%
Dec 15	587	18				< 0.001%
Dec 22	7547	19				0.1%

should have started in late 2017, meaning that this is probably a preliminary scanning campaign by the Satori developers.

7. *August 18 - port 8118*. This port is used by Privoxy, an ad-filtering web proxy. A DoS vulnerability (CVE-2016-1982, CVE-2016-1983) was found at the beginning of 2016. As the anomalies show an increase in *srcIp%* and *srcPort%* along with a decrease in *dstIp%* and *Syn%*, it could be a DDoS attempt.
8. *September 29 - port 3389*. This anomaly corresponds to a scan also detected in [46]. This port is used by Windows Remote Desktop protocol (RDP), a service allowing remote users to connect to another computer through a graphical interface. On this day we have a single IP address targeting several hosts distributed in the entire MAWI network with the same source port. In [46] it is supposed to be either an update issued by an administrator connected to several machines over RDP or an intrusion attempt. We lean towards the intrusion hypothesis as *(a)* several vulnerabilities including arbitrary code execution and denial of service were discovered in 2015 and 2016 (CVE-2015-0079, CVE-2015-2373, CVE-2015-2472, CVE-2016-0019, CVE-2016-0036) and *(b)* it is unlikely for a remote administrator to push an update on a large number of machines over a graphical interface. This anomaly is also detected on October 20.
9. *December 1 - port 5555*. This is an unofficial port used by several services, including Android Debug

Bridge (ADB). Most notably, in 2018 there was a report of the ADB.Miner botnet (i.e., a Mirai variant) infecting Android devices with ADB turned on, but scanning activity on this port was already present since November 2016 [48]. Given the nature of the alert, characterised by a surge in *srcIp%*, *dstIp%* and *srcPort%*, it is most likely an early scan perpetrated by the botnet.

10. *December 22 - port 7547*. This anomaly corresponds to the Mirai variant detected in [8] on December 8. This port is associated to TR-069, an application layer protocol for remote management of customer-provided equipment. In November 2016 a vulnerability was reported on some products using this protocol [49] and few weeks later, several Deutsche Telekom routers suffered an outage due to their injection with this variant.

Thanks to this study, we can provide a tentative false positive rate evaluation. Limiting ourselves to high-relevance anomalies (arbitrarily defined as those with $AS > 15$) in the $N_{min} = 20$ scenario, there are 12 that likely correspond to malicious activity, 5 that potentially correspond to malicious activity and 13 that are likely false positives, resulting in a false positive rate between 43% and 60%. In the case of $N_{min} = 100$, we have 7 anomalies likely related to malicious activity, 3 that are potentially related to malicious activity and 3 that are likely false positives, resulting in a false positive rate between 23% and 46%. In the second case, we filter out some ports, improving the false positive rate, but at the cost of reducing the number of potential attacks detected. Once again, note that the choice of N_{min} affects only the centralized analysis of data collected by the switches, and can be adjusted at any moment without impacting the metric collection process.

Ultimately, as the goal of the proposed IDS is to provide a first line of defense against unknown attacks, to be paired with other, fine-grained methods (e.g., DPI), we care little about the ex-post false positive rate as long as the average daily number of alerts remains compact, and therefore recommend keeping N_{min} low.

6.2.4. Comparison With State-of-the-Art Method [8]

We compare our results with the ones proposed in the original Split-and-Merge work, where a similar analysis is carried on.

The use of approximate computing, made necessary due to hardware and timing limitations, introduces small errors that generally do not impact the AS computation. Yet, there are some key differences between the results of our work and those presented in the original Split-and-Merge works. The reason behind these differences is unclear, as our attempt at reproducing the analysis of the reference work gave results that are consistent with our solution. For this reason, in the following we propose a comparison between the two.

Table 4 shows the total number of anomalies detected over the evaluation period in the two cases $N_{min} = 20$ and $N_{min} = 100$, plus the ones reported in [8], where they use $N_{min} = 20$. Anomalies are grouped according to their AS: (i) low-relevance, with $AS < 10$ (i.e., mean number of anomalies per subnet is less than 1); (ii) medium-relevance, with $10 \leq AS \leq 15$; (iii) high--relevance, with $AS > 15$ As shown in the table, our online version generates a considerably greater amount of alerts, but still not so high to be unmanageable by network administrators.

Table 4: Amount of anomalies detected sorted by relevance.

Relevance	**Original**	$N_{min} = 20$	$N_{min} = 100$
Low (`AS=1-9`)	3001	16289	12104
Medium (`AS=10-15`)	21	263	62
High (`AS=16+`)	8	30	13

To further explore the comparison, let us examine the difference between the most relevant anomalies detected by our solution with the ones reported in the reference work. There are four anomalies that are spotted in both frameworks: the Redis scan on port 6379; the Mirai scan on port 23; the RDP intrusion attempt on port 3389; the Mirai variant on port 7547. Then, there are five anomalies that our system is able to detect but are not reported in [8]: the scan on port 445; the intrusion attempt on port 3128; the stealthy scan on port 52869, potentially linked to the Satori botnet; the DDoS attack on port 8118; the Mirai variant on port 5555. Finally, there are three that we are not able to reproduce despite being detected in [8], namely the Mirai scan on port 2323 and its variants on port 6789 and 23231.

The Mirai scan on port 2323, detected on September 15 with a score of 28 in the reference work, only generates a medium anomaly with AS = 10 on September 8 in our framework (when $N_{min} = 100$), although it is still the port with highest AS of the day. The Mirai variant on port 6789 detected in [8] on December 29 remains undetected by our solution. As the port is mostly unused until December 22, there are not enough data points to

populate feature time series by December 29, making it impossible to calculate an Anomaly Score on five subnets out of nine. A similar situation occurs for the Mirai variant on port 23231 detected in [8] on December 22. The port starts being used in five subnets one week before, and ends up generating only few anomalies in just one subnet, getting filtered out as a local phenomenon.

Interestingly, all these undetected anomalies share in common that the attacked ports were mostly unused until the day of the attack inception. Given that, an administrator may sniff out the attack by looking at the sudden traffic surge (e.g., by giving a greater weight to new-port anomalies) and noticing the major increment of traffic from one week to another.

6.2.5. Detection in 2023

To showcase the relevance of the proposed system to more recent real-world data, we repeat the analysis for the first 7 months of 2023. For the sake of simplicity, we limit ourselves to the $N_{min} = 20$ case, and for each day, we only analyze ports that are interested by at least N_{min} packets in at least two subnets. As a consequence, we observe a massive reduction in low-relevance anomalies, and notably, the disappearance of events with $AS = 1$, as shown in Figure 14.

As we did for the 2016 period, we now briefly discuss the most notable anomalies detected in 2023. Note that all of these anomalies go undetected in MAWILabs reports.

1. *April 13 - port 52869*. Yet another anomaly on port 52869, used by the UPnP protocol suite. Port 52869 is often victim of scanning activity related to the Satori botnet, such as the one we spotted for the 2016 period. In this case, there is an increase in *srcIp%*, *srcPort%*, *meanSize*, and *stdSize*, alongside a decrease in *Syn%*. Given the increased packet size and the prevalence of non-SYN packets, this does not look like a scan. It is more likely a DDoS attack, exploiting UPnP devices to amplify TCP traffic. This can be accomplished by sending SOAP requests to UPnP devices that were previously identified, spoofing the target's address. The UPnP devices then respond by sending the HTTP location of their XML description file to the target IP address, potentially saturating its bandwidth.
2. *May 4 and July 6 - port 7574*. Port 7574 is used by Oracle Coherence, a distributed cache and in-memory data grid system. The port behaviour exhibits all the traits of a DDoS attempt: surge in *srcIp%* and *srcPort%*, as multiple connections are being established, drop in *dstIp%*, as Oracle servers are being specifically targeted, increase in *meanSize* and decrease in *stdSize*, as big and similar packets are being crafted, and finally a reduction in *Syn%*. Some vulnerabilities were disclosed in 2022, leading to DoS (CVE-2022-21570) and system takeover (CVE-2022-21420). Most notably, on June 20, i.e., seven weeks after the first attack and two weeks before the second attack, a vulnerability was discovered for a component of Oracle Coherence called Netty (CVE-2023-34462). Due to this vulnerability, a low privileged attacker can establish TCP connections with an Oracle Coherence server and allocate 16MB of heap memory per connection, potentially causing a Denial of Service if no idle timeout handler is configured.

To summarize the outcomes of the two evaluations, we observe that in 2016 we have almost one high-relevance anomaly per day, while in 2023 we have approximately one relevant anomaly every 5 days. We must consider that an apples-to-apples comparison between the two analyses is of little interest, as the number of attacks in a given period (hence, the amount of detected anomalies) can vary greatly, depending on factors such as the vulnerabilities discovered in the recent time. Consequently, the main message that we can draw is that port-based approaches have been proven over time to be able to detect malicious activity such as port scanning, botnet spreading and DDoS attacks, and that the proposed architecture is able to execute metric collection directly on the data plane, enabling line rate analysis of large volumes of data without requiring offline processing.

6.3. Scalability

In this section we discuss possible scalability issues related to the metric collection process. Specifically, we want to consider the potential impact of unusually large traffic volumes. Looking at the MAWI dataset, we can observe that the number of TCP packets per subnetwork in the 15-minute traces is in the order of tens of millions. We also observe that the percentage of traffic on a given port is usually below 1% and hardly ever beyond 4%. Let us suppose a very pessimistic case in which the number of packets traversing a single switch within a time slot is 100 million and the port under analysis accounts for 5% of the traffic, that is 5 million packets. Additionally, let us suppose that every packet has unique source and destination IP addresses, such that the IP address cardinalities to estimate are also 5 million.

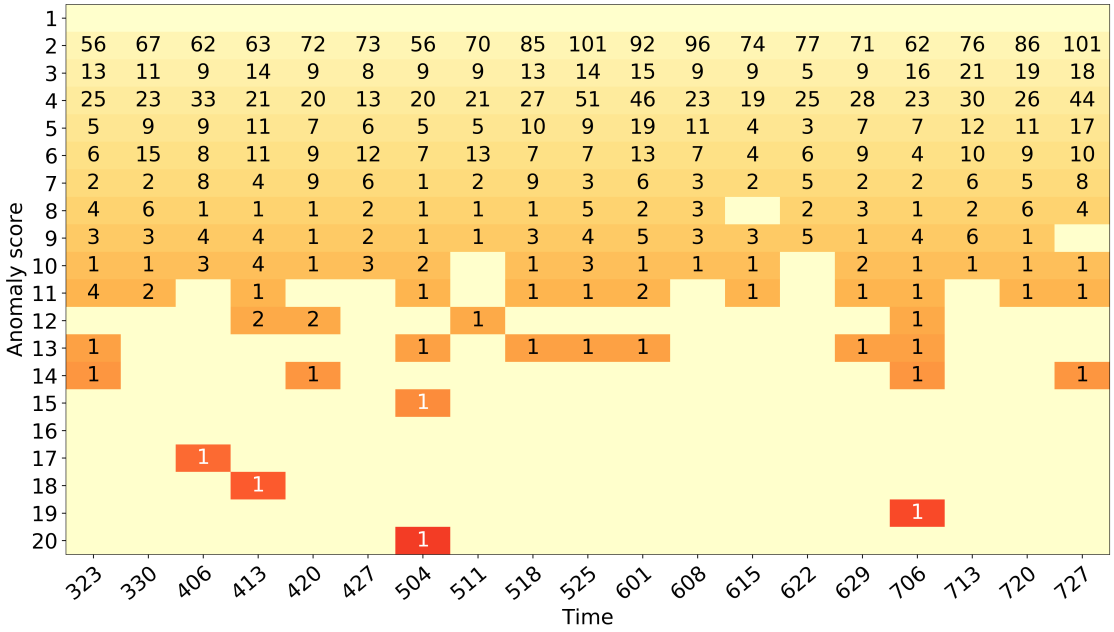


Figure 14: Anomaly Scores for the 2023 period

In the Welford extern, we use 24-bit registers for the number of packets, the number of SYN packets and the average packet size, and a 40-bit register for M_2. 24 bits allow us to count packets up to 16.7 million, so there is no risk of overflow. The average packet size cannot overflow either, as it is bounded by the maximum size of IP packets. Indeed, if the value of *avg* grows excessively large, such as following a sequence of large-sized packets, it is probable that the next packet sizes observed will be smaller than *avg*. Therefore, next increments to *avg* will likely be negative, pushing its value towards the actual average. Finally, the M_2 register is always incremented by the product between the two deltas, which is bounded (as deltas are bounded) but almost always positive (as deltas are likely of the same sign). Being the increment always positive, there is a risk of overflow in case of large traffic volumes, and that is why we conservatively use 40 bits instead of 24 for M_2.

As specified in Section 4, the HLL extern can estimate cardinalities up to 6 million. Therefore, even in this very pessimistic scenario where every packet has unique IP addresses, there is no risk of saturating the HLL buckets.

Ultimately, the proposed design is dimensioned after the expected data rate, and it could need to be resized if employed in a different use case. It must be observed, in fact, that 100 million packets per 15-minutes is far lower than the maximum throughput of a NetFPGA Sume switch, that is in the order of tens of billions of packets. On the other hand, note that we use 15-minute time slots just because MAWI traces come in that format. A possible approach could be to adjust the length of the observation window while maintaining a constant number of observed packets. On top of that, the centralized controller may increase the frequency of metric queries from one at the end of each time slot to more than one, spread out over time. This way, eventual overflows and bucket saturation effects may be spotted, and correction strategies could be employed.

For example, if a packet count overflow is detected, it can be easily compensated by the centralised controller by just summing the maximum allowed value in the related register (i.e., $2^{24} - 1$) to the last count retrieved.

As another example, intermediate queries of packet standard deviation can be retained and used instead of the end-of-time-slot one, as the expected estimation error grows as the number of observed packets increases (as shown in Section 6.1.2). In fact, assuming the packet size distribution to be normal (as proven in [8]), we don't need to compute the standard deviation using all the packets observed in a given time slot, but we can limit to a subset based on the confidence interval we are willing to tolerate. To calculate this, we can use the following formula from [50]:

$$Pr\left(ST\sqrt{\frac{N-1}{q_{1-\frac{\alpha}{2}}}} < \sigma < ST\sqrt{\frac{N-1}{q_{\frac{\alpha}{2}}}}\right) = 1-\alpha \quad (5)$$

where N is the number of packets used for the estimation, σ is the real standard deviation, ST is the estimated standard deviation, q_p is the p-th quantile of the chi-square distribution with $N-1$ degrees of freedom, and $\alpha - 1$ is the confidence interval. For example, we could retain an intermediate ST query with $N = 1000$ and have a 95% probability that the real standard deviation σ falls within the interval $[0.96 \cdot ST, 1.05 \cdot ST]$.

6.4. Resource Usage

Table 5 shows the resources required to synthesize two variants of the proposed design. The first version has a simplified Routing Engine, where data plane traffic is only routed according to a dst IP table, while the second version employs the full routing logic as described in Section 5.

The usage of LUTs, BRAM modules and Flip-Flops increases by 11.61%, 18.01% and 31.11% respectively, showing a clear bottleneck on FF availability. This prevents us from upscaling currently integrated modules or adding further logic to our solution. Rather than the introduction of the additional extern modules, the main driver of the bottleneck is the increased complexity of the P4 pipeline, which accounts for around 80% of the FF usage growth, hinting that the bottleneck could be reduced by (i) implementing part of the routing engine in a dedicated extern module to ease the P4 pipeline and (ii) merging some of the SBI messages to decrease the redundancy of the Parser and the Deparser.

Table 5: NetFPGA Resource Usage.

Design	LUT	BRAM	FF
Split-and-Merge + Simple Routing Engine	35.08%	51.87%	47.75%
Split-and-Merge + Full Routing Engine	46.69%	69.86%	78.86%

7. Conclusions

In this paper we propose a CIDS-like unsupervised distributed anomaly detection system. For this purpose we refine the Split-and-Merge algorithm and design NetFPGA board forwarding system to act as both programmable switches and network probes, managed by a remote controller that, via a dedicated new South-Bound-Interface we designed, retrieves the metrics from the data plane and uses them to detect anomalies and traffic shifts. We leverage approximation and data sketches to cope with the resource limitations and the need to perform operations at line rate. On top of anomaly detection, we also allow network administrators to enforce basic mitigation actions, such as traffic redirecting, port blocking and IP allowlisting or blocklisting.

We test our solution against the MAWI Archive dataset, showing how it can catch several intrusion attempts that actually took place in 2016, to compare our work with the reference algorithm, and in 2023, to showcase the effectiveness of the approach in a more recent time, generating few high-level anomalies per day thanks to the false positive filtering granted by the collaborative approach.

The current practical shortcoming of our solution is its scaling with multiple monitored port. On one hand, the evaluation from both our work and [8, 46] shows that monitoring 15 minutes per week is enough to catch network level traffic shifts, and therefore the controller could assign each 15-minute time slot to a different port to scale with a wide span of ports every week. Nevertheless, we are currently working on a solution to monitor several ports at the same time, as this would *(a)* increase the number of monitored ports per day, *(b)* provide more freedom to customize the time slot duration and *(c)* increase the flexibility in terms of which ports to analyse without reducing the coverage. This effort could also support the introduction of additional feature computing at line rate.

Besides, established the unfeasibility of monitoring every port, apart from *how many* ports to analyse, an important open question is *which* ones. A potential answer that we are exploring is to use a heavy hitter detection algorithm to figure out the most used ports, while possibly leaving some room for monitoring random or arbitrary ones.

Acknowledgements

This work was funded by the H2020 AI@EDGE project (https://aiatedge.eu; contract nb: 101015922), the French ANR HEIDIS project (https://heidis.roc.cnam.fr; contract nb: ANR-21-CE25-0019), and the IPCEI ME/CT Orange project (contract nb: DOS0239248/00).

References

[1] H. Song, T. Sproull, M. Attig, J. Lockwood, Snort offloader: A reconfigurable hardware nids filter, in: International Conference

on Field Programmable Logic and Applications, 2005., IEEE, 2005, pp. 493–498.

[2] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, J. Sherry, Achieving 100gbps intrusion prevention on a single server, in: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), 2020, pp. 1083–1100.

[3] M. Ceška, V. Havlena, L. Holík, J. Korenek, O. Lengál, D. Matoušek, J. Matoušek, J. Semric, T. Vojnar, Deep packet inspection in fpgas via approximate nondeterministic automata, in: 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2019, pp. 109–117.

[4] R. Ricart-Sanchez, P. Malagon, J. M. Alcaraz-Calero, Q. Wang, Netfpga-based firewall solution for 5g multi-tenant architectures, in: 2019 IEEE International Conference on Edge Computing (EDGE), IEEE, 2019, pp. 132–136.

[5] Y.-K. Lai, P.-Y. Huang, H.-P. Lee, C.-L. Tsai, C.-S. Chang, M. H. Nguyen, Y.-J. Lin, T.-L. Liu, J. H. Chen, Real-time ddos attack detection using sketch-based entropy estimation on the netfpga sume platform, in: 2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), IEEE, 2020, pp. 1566–1570.

[6] Y.-K. Lai, K.-P. Chang, X.-W. Ku, H.-L. Hua, A machine learning accelerator for ddos attack detection and classification on fpga, in: 2022 19th International SoC Design Conference (ISOCC), IEEE, 2022, pp. 181–182.

[7] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, A. Madeira, Flowlens: Enabling efficient flow classification for ml-based network security applications., in: NDSS, 2021.

[8] A. Blaise, M. Bouet, V. Conan, S. Secci, Detection of zero-day attacks: An unsupervised port-based approach, Computer Networks 180 (2020) 107391.

[9] P. Flajolet, É. Fusy, O. Gandouet, F. Meunier, Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm, in: Discrete Mathematics and Theoretical Computer Science, Discrete Mathematics and Theoretical Computer Science, 2007, pp. 137–156.

[10] B. Welford, Note on a method for calculating corrected sums of squares and products, Technometrics 4 (3) (1962) 419–420.

[11] A. A. Efanov, S. A. Ivliev, A. G. Shagraev, Welford's algorithm for weighted statistics, in: 2021 3rd International Youth Conference on Radio Electronics, Electrical and Power Engineering (REEPE), IEEE, 2021, pp. 1–5.

[12] P4-netfpga-split-and-merge, "https://github.com/cedric-cnam/P4-NetFPGA-Split-and-Merge/" (2023).

[13] MAWI-Working-Group, Mawi traffic archive, "https://mawi.wide.ad.jp/mawi/" (2023).

[14] R. Fontugne, P. Borgnat, P. Abry, K. Fukuda, Mawilab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking, in: Proceedings of the 6th International COnference, 2010, pp. 1–12.

[15] I. Sharafaldin, A. H. Lashkari, S. Hakak, A. A. Ghorbani, Developing realistic distributed denial of service (ddos) attack dataset and taxonomy, in: 2019 International Carnahan Conference on Security Technology (ICCST), IEEE, 2019, pp. 1–8.

[16] Y. Xing, H. Shu, H. Zhao, D. Li, L. Guo, Survey on botnet detection techniques: Classification, methods, and evaluation, Mathematical Problems in Engineering 2021 (2021) 1–24.

[17] M. Ahmed, A. N. Mahmood, J. Hu, A survey of network anomaly detection techniques, Journal of Network and Computer Applications 60 (2016) 19–31.

[18] Snort, "https://www.snort.org/" (2003).

[19] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, J. Ucles, Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification, in: Proc. IEEE Workshop on Information Assurance and Security, Vol. 85, 2001, p. 90.

[20] G. Gu, J. Zhang, W. Lee, Botsniffer: Detecting botnet command and control channels in network traffic, in: 16th Annual Network & Distributed System Security Symposium, 2008, p. 19.

[21] W. Wang, Y. Shang, Y. He, Y. Li, J. Liu, Botmark: Automated botnet detection with hybrid analysis of flow-based and graph-based traffic behaviors, Information Sciences 511 (2020) 284–296.

[22] A. M. Araujo, A. B. de Neira, M. Nogueira, Autonomous machine learning for early bot detection in the internet of things, Digital Communications and Networks (2022).

[23] K. Wu, Z. Chen, W. Li, A novel intrusion detection model for a massive network using convolutional neural networks, Ieee Access 6 (2018) 50850–50859.

[24] H. Yang, F. Wang, Wireless network intrusion detection based on improved convolutional neural network, Ieee Access 7 (2019) 64366–64374.

[25] S.-C. Chen, Y.-R. Chen, W.-G. Tzeng, Effective botnet detection through neural networks on convolutional features, in: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), IEEE, 2018, pp. 372–378.

[26] J. Kim, Y. Kim, V. Yegneswaran, P. Porras, S. Shin, T. Park, Extended data plane architecture for in-network security services in software-defined networks, Computers & Security 124 (2023) 102976.

[27] B. Iglewicz, D. Hoaglin, The asqc basic references in quality control: statistical techniques, How to detect and handle outliers 16 (1993) 1–87.

[28] HyperLogLog++ functions | BigQuery.
URL `https://cloud.google.com/bigquery/docs/reference/standard-sql/hll_functions`

[29] R. Harrison, Q. Cai, A. Gupta, J. Rexford, Network-wide heavy hitter detection with commodity switches, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.

[30] Y. Chabchoub, R. Chiky, B. Dogan, How can sliding hyperloglog and ewma detect port scan attacks in ip traffic?, EURASIP Journal on Information Security 2014 (2014) 1–11.

[31] K.-Y. Whang, B. T. Vander-Zanden, H. M. Taylor, A linear-time probabilistic counting algorithm for database applications, ACM Transactions on Database Systems (TODS) 15 (2) (1990) 208–229.

[32] M. Thill, W. Konen, T. Bäck, Online anomaly detection on the webscope s5 dataset: A comparative study, in: 2017 Evolving and Adaptive Intelligent Systems (EAIS), IEEE, 2017, pp. 1–8.

[33] D. O'Shea, V. C. Emeakaroha, J. Pendlebury, N. Cafferkey, J. P. Morrison, T. Lynn, A wavelet-inspired anomaly detection framework for cloud platforms., in: CLOSER (1), 2016, pp. 106–117.

[34] S. Sahmoud, H. R. Topcuoglu, Dynamic multi-objective evolutionary algorithms in noisy environments, Information Sciences 634 (2023) 650–664.

[35] S. Ibanez, G. Brebner, N. McKeown, N. Zilberman, The p4-netfpga workflow for line-rate packet processing, in: Proc. of the 2019 ACM/SIGDA Intern. Symposium on FPGAs, 2019, pp. 1–9.

[36] P4-netfpga, "https://github.com/NetFPGA/P4-NetFPGA-public/wiki" (2017).

[37] A. Kulkarni, M. Chiosa, T. B. Preußer, K. Kara, D. Sidler, G. Alonso, Hyperloglog sketch acceleration on fpga, in: 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), IEEE, 2020, pp. 47–56.

[38] Xilinx, 7 series dsp48e1 slice, "https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1 (2018).
[39] A. T.-Y. Chen, R. Gupta, A. Borzenko, K. I.-K. Wang, M. Biglari-Abhari, Accelerating superbe with hardware/software co-design, Journal of Imaging 4 (10) (2018) 122.
[40] T. Granlund, P. L. Montgomery, Division by invariant integers using multiplication, in: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation, 1994, pp. 61–72.
[41] D. G. Bailey, M. J. Klaiber, Efficient hardware calculation of running statistics, in: 2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013), IEEE, 2013, pp. 196–201.
[42] P. A. W. Group, P4runtime specification, version 1.3.0, "https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html" (2021).
[43] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM CCR 38 (2) (2008) 69–74.
[44] M. Patetta, S. Secci, S. Taktak, A lightweight southbound interface for standalone p4-netfpga smartnics, in: 2022 1st International Conference on 6G Networking (6GNet), IEEE, 2022, pp. 1–4.
[45] J. Mazel, R. Fontugne, K. Fukuda, A taxonomy of anomalies in backbone network traffic, in: Proceedings of 5th International Workshop on TRaffic Analysis and Characterization, TRAC 2014, 2014, pp. 30–36.
[46] A. Blaise, M. Bouet, S. Secci, V. Conan, Split-and-merge: detecting unknown botnets, in: 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, 2019, pp. 153–161.
[47] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th {USENIX} security symposium ({USENIX} Security 17), 2017, pp. 1093–1110.
[48] 360netlab mirai scanner, "https://data.netlab.360.com/mirai-scanner/" (2016).
[49] Eir's d1000 modem is wide open to being hacked, "https://devicereversing.wordpress.com/2016/11/07/eirs-d1000-modem-is-wide-open-to-being-hacked/" (2016).
[50] D. J. Sheskin, Handbook of parametric and nonparametric statistical procedures, Chapman and hall/CRC, 2003.