Portest: Port Scan Detection on Non-Programmable Switches using TCAM and Randomized Algorithm

Timon Krack and Martina Zitterbart, KIT
Institute of Telematics, Karlsruhe Institute of Technology (KIT)
{timon.krack,martina.zitterbart}@kit.edu

ABSTRACT

Monitoring network traffic for detecting security events is crucial for the effective operation of intrusion detection systems (IDS). While programmable switches offer the flexibility to execute monitoring algorithms directly in the data plane, non-programmable switches lack such capabilities and traffic needs to be mirrored and processed externally, leading to scalability and performance challenges. In this paper, we present *Portest*, a novel algorithm that enables the detection of port scans on non-programmable switches without mirroring traffic. Portest installs a constant number of flow rules with specific stochastic properties in the Ternary Content Addressable Memory (TCAM) of the switch and uses the match counter values for detection. Our results demonstrate that Portest can efficiently detect real-world port scans on non-programmable hardware.

1 INTRODUCTION

Monitoring network traffic for detecting security events such as port scans is a key component of intrusion detection systems (IDS) and is highly important for ensuring the safe operation of network infrastructures. Usually, port scans are detected by algorithms running on an external server that receives mirrored traffic from a switch [6, 16]. Advanced programmable switches support the execution of code directly in the data plane, for instance by using the P4 programming language [5, 14]. They enable IDS to run monitoring algorithms directly in the networking devices, which increases scalability and decreases the overhead of mirroring and processing traffic externally. However, switches that do not support a programmable data plane (non-programmable switches) are commonly used, and are significantly cheaper. Their functionality is often limited to installing match/action rulesets that the switch applies to incoming packets. It is not possible to execute more complex instructions or algorithms on the switch, instead traffic has to be processed externally.

 ACM ISBN 979-8-4007-2087-1/25/09. https://doi.org/10.1145/3748496.3748988

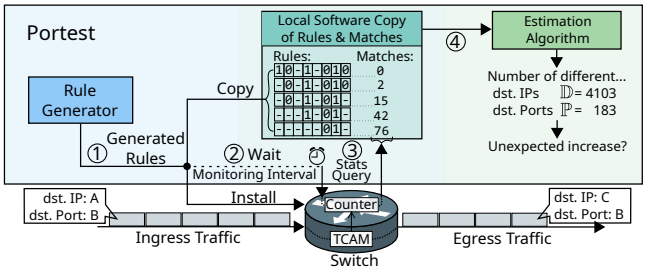

Figure 1: Overview of Portest's Architecture

Existing monitoring and IDS techniques that run on non-programmable switches are highly constrained and often lack scalability due to the hardware limitations [3, 11, 13].

This paper presents *Portest*, a novel monitoring algorithm running on non-programmable switches that detects horizontal and vertical port scans. It can also run on programmable switches to offload packet processing resources to the TCAM. Portest runs a lightweight application on an external system (e.g., SDN controller), and can manage many switches simultaneously with very low overhead.

An architectural overview of Portest is shown in Figure 1. In the figure, only a single switch is shown for simplicity. First, a set of flow rules is installed on the switch ①. Each rule matches packets with multiple, random IP destination addresses and/or layer 4 destination ports. The rules have specific stochastic properties that allow for the subsequent extraction of monitoring information. Portest assumes that the switch tracks match counters for the flow rules, which is a common feature even on low-end hardware. After a monitoring interval expires ② (a few seconds), these match counters are queried ③. Portest evaluates the received counter values and estimates the number of different destination IP addresses $\mathbb{D}$ and the coverage of used destination ports $\mathbb{P}$. Port scans are detected by unexpected increases in these metrics ④. The port scan detection is based only on the match counter values. No traffic is transferred out of the switch and no per-flow memory allocation is made, making the approach highly scalable. The number of occupied TCAM rules is constant. Since all rules are matched in parallel, the packet processing overhead is also constant. When a port scan is detected, traffic from just the affected switch can be mirrored and analyzed in more detail to take mitigating actions.

This paper presents Portest and evaluates results with real-world benign network traffic and real hardware. The key contributions include:

- Novel approach for metric estimation and detecting security events on *non-programmable* commodity switches
- Illustration of Portest's use of TCAM rules
- Evaluation using traffic datasets containing real-world background traffic and self-generated port scans
- Detection of real-world port scan in an ISP traffic dataset

2 RELATED WORK

Port scan detection is an important and widely explored topic in the area of network monitoring. Related projects perform port scan detection in the context of software defined networks (SDN) using various approaches. They can be grouped into three categories, none of which apply to Portest.

First, approaches that assume an SDN context where the controller installs per-flow forwarding rules. When a packet for a new flow arrives, the switch sends a Packet-In message to the SDN controller, which can be analyzed to perform port scan detection [13]. By keeping track of per-flow rate counters for different addresses, detections are made using a threshold-based decision. Two other projects query the per-flow rule statistics to detect flows that transfer just a single packet, which is often the case during a port scan [3, 11]. The per-flow rule statistics can also be used for a machine-learning based detection [2]. However, per-flow forwarding rules are not scalable in larger networks due to the high number of flows and limited rule table capacity.

Second, multiple approaches use algorithms to detect port scans in traffic mirrored to an external location, like a monitoring server. One project developed an efficient method to distribute and parallelize the processing of the mirrored traffic to detect port scans [6]. Similarly, the amount of traffic that needs to be processed can be reduced by using traffic sampling and using a reduced sampling rate for elephant flows, and then detecting port scans in the sampled traffic [16]. PD-CPS [17] also mirrors traffic to a monitoring server, and opposed to Portest, focuses on detecting stealthy port scans operating at an extremely low speed, which can be ongoing for months. While the processing of mirrored traffic on external monitoring servers provides great flexibility for the detection algorithm, the network and hardware utilization overhead is large and increases with higher traffic rates since the packets need to be copied. The network overhead from mirroring can be prevented by running the port scan detection directly on end systems, for instance, by capturing connection attempts using eBPF [10]. However, this is only possible if software can be installed on the end systems.

Third, to prevent both network and hardware overhead, many approaches implement the port scan detection directly on switches with a programmable data plane, for instance, using the P4 programming language [1, 14]. Using P4, code can be executed for each received packet directly on the switch, preventing the need to mirror and process the traffic externally. In particular, port scan detection was implemented using P4 by estimating the number of concurrently active flows, which increases sharply when a port scan attempts to open many new connections to different destinations [5]. Portest uses a similar approach as part of its detection strategy, however uses a different method to obtain estimations.

3 PORT SCAN DETECTION

This section explains Portest's port scan detection strategy and its implementation on non-programmable hardware.

3.1 Basic Approach

Portest concurrently estimates two metrics for each monitoring interval (a few seconds) directly on non-programmable switches, without a per-flow memory allocation and without mirroring traffic to an external server. First, the destination cardinality $\mathbb{D}$, equal to the number of distinct destination IP addresses that have received at least one packet in the current monitoring interval. Second, the destination port coverage $\mathbb{P}$, quantifying the amount of ports between $[1, 1024]$ which have received at least one packet for any destination IP address in the current monitoring interval. Only lower port numbers are considered to prevent counting local ports.

Both metrics are tracked over time to detect port scans. In a *horizontal* port scan, the attacker targets one or few ports that get scanned across a large number of different destination IP addresses, which increases $\mathbb{D}$. In a *vertical* port scan, few IP addresses are scanned but on many different destination ports, which increases $\mathbb{P}$. A similar detection strategy for port scans based on cardinality estimations is described for a P4-based implementation in [5].

Portest's estimation algorithm is inspired by the HyperLogLog (HLL) cardinality estimation algorithm [7]. HLL computes a uniform hash function over each element and produces an estimation based on counting the maximum number of leading zeros in one of the hashes. Both operations are not possible on non-programmable switches. Portest leverages the properties of TCAM to enable a stochastical cardinality estimation to determine $\mathbb{D}$ and $\mathbb{P}$.

3.2 Metric Estimation

A constant number of flow rules are installed in the TCAM. They use the IP destination address and the destination port match fields, and count the number of matching packets. The TCAM stores *ternary* bitmask for match fields, i.e., each bit can be one of three values (`0`, `1` or "don't care" (`-`)). Portest uses randomized bitmasks to probabilistically match packets.

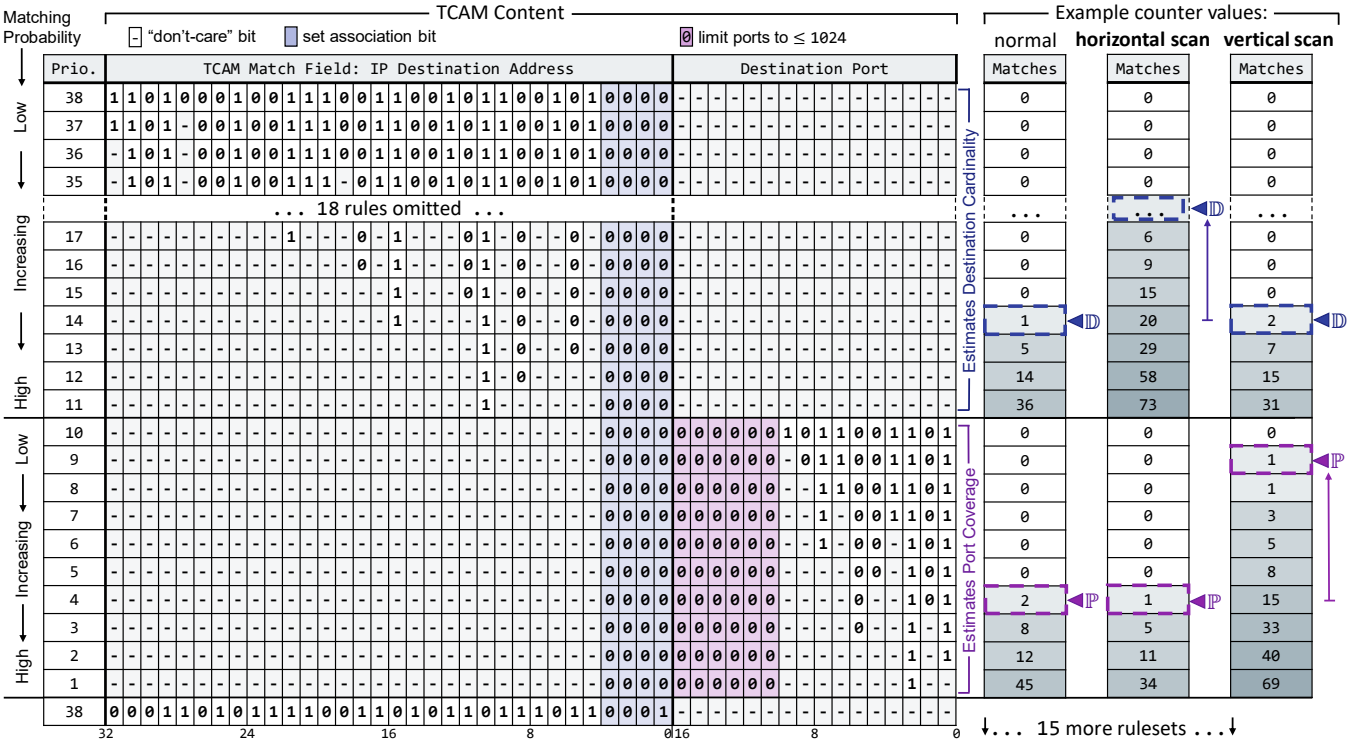

Figure 2: Example of the rule structure installed in the TCAM for estimating $\mathbb{D}$ and $\mathbb{P}$. Left side: TCAM entries in binary. Right side: Example match counters for traffic with and without a vertical/horizontal port scan.

Stochastically, if more "don't care" bits are contained in the bitmask, the probability that a packet will match the rule increases. Portest uses this property to build specially designed collections of rules ("*rulesets*") as shown on the left side of Figure 2. The first rule of the ruleset (at the top) matches a full, randomly generated bitmask for the packet IP destination address. The probability that there is a packet in the ingress traffic with exactly the corresponding IP destination address is very low. For each next rule added to the ruleset, a bit is randomly chosen and replaced with the "don't care" state for all further rules. This doubles the probability per packet that the rule matches. After adding more rules with the same scheme, all bits eventually got replaced by the "don't care" state. The probability that a packet matches the rules with many "don't care" bits is much higher.

The last s bits (in this example: $s = 4$) never get replaced by "don't care" bits. They are instead used to separate the ingress traffic into $2^s = 16$ sets by their destination IP address. The first set, which is shown in the figure, only matches packets with a destination address ending in `0000`. The next set (first rule visible at the bottom of the figure) uses `0001` to handle another $\frac{1}{16}$th of the traffic, and so on.

The key thought of the TCAM rule design is based on the fact that packets with the same IP destination address are always matched by the same rule. By basing the estimation only on whether a rule has been matched at least once or not at all, a large flow sending hundreds of packets per second counts exactly the same as a flow with a single packet. When a security event such as a horizontal port scan occurs, many individual packets are sent to different destination addresses. With an increased number of distinct destination addresses, it becomes more likely that the rules with fewer don't care bits (and therefore, lower matching probability) also receive some matches. To estimate the destination cardinality $\mathbb{D}$, Portest finds the rule with the least "don't care bits" d (i.e., smallest matching probability) that has still received at least one match, then averages across all 2^s rulesets. Assuming 32-bit wide IPv4 addresses, the estimation is derived from $\mathbb{D} = 2^{32-d} \cdot \beta$, where β is a bias correction term. Details of β and an analysis of an equivalent cardinality estimation strategy for *destination* addresses can be found in [8], which shows the estimation deviates $< 5\%$ with 4000 TCAM rules.

The port coverage $\mathbb{P}$ is estimated with an analogous strategy using the destination port match field. To prevent counting local ports from response packets sent back to connection initiators, only ports below $2^{10} = 1024$ are considered by always setting the leftmost $16 - 10 = 6$ bits of the destination port match fields to `0` (colored in purple). The range of considered ports can be adapted by changing the number of leftmost `0`-bits.

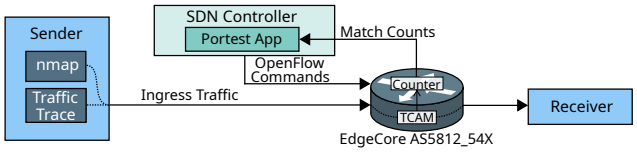

Figure 3: Overview of testbed setup for Portest

3.3 Port Scan Detection

Using the estimations of $\mathbb{D}$ and $\mathbb{P}$, Portest tracks their changes over time to detect port scans. Slow changes within longer periods of time are expected as the total traffic volume changes, while fast changes without a simultaneous change in the traffic volume are unexpected and get detected. Examples of the changes in the match counters and the increased estimation are shown on the right side of Figure 2, for normal traffic (left column), traffic with a horizontal port scan (middle column) and a vertical port scan (right column). If a horizontal port scan begins, the number of distinct destination IP addresses occurring in the ingress traffic increases suddenly, which causes the estimation of $\mathbb{D}$ to increase. Similarly, a vertical port scan increases the number of distinct destination ports, which increases $\mathbb{P}$.

A horizontal (or vertical) port scan detection is triggered if the difference between the estimation of $\mathbb{D}$ (or $\mathbb{P}$) compared to the N-minute rolling average of the previous values is greater than the detection limit of t percent. While t controls the minimum magnitude of a spike to trigger a detection, increasing N ensures that legitimate traffic pattern changes happening over time do not trigger a false detection. The parameter selection depends on the estimation accuracy, which is better for larger TCAM sizes. For small TCAMs, t should be chosen higher to prevent false positive detections due to estimation outliers. Only triggering a port scan detection if two consecutive estimations are elevated additionally prevents false positive decisions from individual outliers in the estimations.

Since the port scan detection is based on a relative change compared to the legitimate traffic, it is beneficial to exclude traffic that is guaranteed to not be part of a port scan. If there is less traffic counted overall, relative changes are bigger and are detectable for smaller port scans. An additional rule is added with higher priority than all other rules. It matches TCP packets that have the ACK flag enabled, but the SYN, RST, and FIN flags disabled. The bits for the other flags are set to the "don't care" state. Therefore, the rule only matches packets that are sent in already established TCP connections. For these packets, the additional rule matches with higher priority, which effectively excludes them since the other rules for estimating $\mathbb{D}$ and $\mathbb{P}$ are not updated. Note that this rule is not shown in Figure 2 for simplicity.

4 EVALUATION

Portest is evaluated using a testbed setup. Multiple scenarios are considered, including vertical and horizontal port scans at different intensities of background traffic.

4.1 Testbed Setup and Datasets

For evaluating Portest, a testbed setup is used to replay real-world traffic captures and to send port scans. An overview of the testbed setup is shown in Figure 3. There is a sending and receiving server, connected to an EdgeCore AS5812_54X hardware switch. The sending server replays a traffic trace, and can also send port scan traffic by running the `nmap` tool [9]. The switch is controlled using OpenFlow [12] from a server acting as the SDN controller. It runs the Portest SDN-App, which adds flow rules and reads the match counters. Portest does not depend on OpenFlow and can be implemented with different protocols or platforms.

The switch available for the evaluation is a comparatively old and weak model with very limited TCAM capacity of < 1000 rules. Switches with TCAM capacities of tens of thousands and more are common nowadays. Running Portest on a switch with a larger TCAM would produce more accurate results, as more rulesets can be used. Nevertheless, even the limited TCAM capacity is sufficient to perform the port scan detection. In these experiments, a monitoring interval of 3 seconds, $t = 20\%$ and $N = 60$ seconds are selected.

Portest is evaluated using two datasets, containing traffic at significantly different scales. The CICIDS-2017 dataset [15] 'Monday'-Trace contains legitimate background traffic of ≈ 10.000 devices over the course of one day. For evaluation, a five minute long segment from 12:51 to 12:56 UTC is chosen, because it contains the most activity. The CAIDA OC48 Peering Point Traces [4] contains a capture of one hour of anonymized ISP traffic with ≈ 4000 active devices in one second. A five minute long segment from capture points `A` and `B` are used, from 07:00 to 07:05 UTC. The traffic from point `A` contains a real-world horizontal port-scan, which Portest correctly identifies.

4.2 Horizontal and Vertical Port Scans

This experiment evaluates how Portest reacts to horizontal and vertical port scans by observing the obtained estimations for $\mathbb{D}$ and $\mathbb{P}$ over time. The CICIDS-2017 dataset is replayed from the sending server and sent through the switch. After 60 seconds, a horizontal port scan is launched by `nmap` on the sending server. It is configured to scan for open SSH ports within a `/16` subnet. The speed (and therefore, stealthiness) of an `nmap` scan can be configured by timing profiles, where `T3` is the default speed and `T5` is the most aggressive. This scan is executed with the more stealthier profile `T2`. After

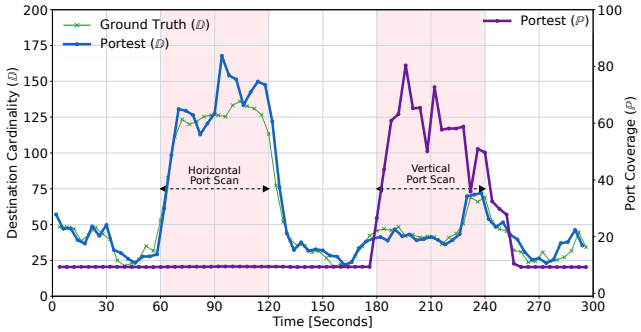

Figure 4: Portest's Estimations of $\mathbb{D}$ and $\mathbb{P}$ during horizontal and vertical port scan, with legitimate background traffic from CICIDS-2017

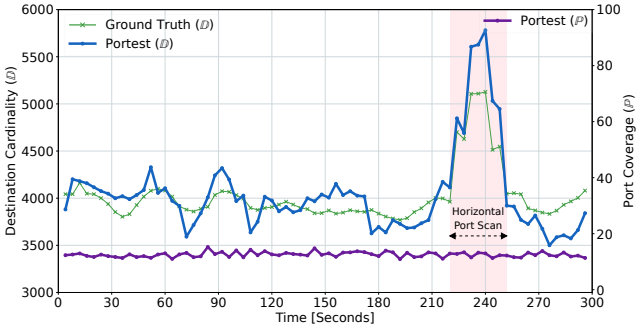

Figure 5: Portest detects real-world horizontal port scan in CAIDA ISP traffic capture

60 seconds, `nmap` is stopped. Then, at 180 seconds, `nmap` is started again, running a vertical port scan for 60 seconds.

Figure 4 shows the estimations for the destination cardinality $\mathbb{D}$ (blue, left y-axis) and for the port coverage $\mathbb{P}$ (purple, right y-axis). The ground truth values for $\mathbb{D}$ are also shown, determined by counting the exact number of different destination addresses in a capture of the transmitted traffic using a hashtable. For $\mathbb{P}$, no ground truth values are available: the metric is only determined from traffic not already matched by one of the higher-priority rules used for estimating $\mathbb{D}$. Therefore, determining the expected value would imply simulating the previous behavior of the TCAM rules. However, this does not make a difference for ensuring relative changes are correctly detected during a scan. The x-axis shows the time in seconds since the start of the traffic replay.

Findings: Portest accurately estimates the destination cardinality $\mathbb{D}$, as the estimation consistently remains close to the ground truth value. When the horizontal port scan starts, the increase in $\mathbb{D}$ is immediately captured and the port scan gets detected. Similarly, when the vertical port scan starts, Portest quickly increases its estimate of $\mathbb{P}$. Both security events were correctly detected and classified.

The experiment was repeated 150 times. Both port scans were successfully detected in 97.3% of the runs. In the other 2.7% of the runs, the estimation of $\mathbb{D}$ (or $\mathbb{P}$) was undershooting at the same time the horizontal (or vertical) port scan started, which prevented triggering the detection. In these cases, the scans were also successfully detected, however, delayed by an additional monitoring interval (i.e., 3 seconds).

4.3 Horizontal Scan in ISP-Traffic

In this experiment, Portest is executed while the sending servers replays the CAIDA traffic capture containing ISP traffic. The traffic volume is > 100 times higher compared to the previous experiment. The resulting estimations of $\mathbb{D}$ (blue, left y-axis) and $\mathbb{P}$ (purple, right y-axis) are shown over time (x-axis) in Figure 5. The traffic contains significantly more active devices with $\mathbb{D} \approx 4000$. Even though no `nmap` scan is launched, a port scan is detected after ≈ 220 seconds into the 5-minute segment of the dataset, corresponding to 07:03:50 UTC in the capture file. The port coverage $\mathbb{P}$ remains constant with a few smaller deviations.

Findings: The estimations of $\mathbb{D}$ remain close to the ground truth values, using the same, constant number of TCAM rules as in the previous experiment. Investigating the dataset traffic around the detected timestamp confirms a horizontal port scan executed by the IP address `225.242.204.167` against many IP addresses with similar prefixes. Therefore, Portest correctly detected and classified the real-world port scan captured in the dataset.

4.4 Vertical nmap-Scan with ISP-Traffic

This experiment generates a vertical port scan with `nmap` while replaying the ISP traffic. Traffic from the same 5 minute timespan, but from monitoring point `B` is used (which does not contain the vertical port scan). The `nmap`-scan is started 120 seconds into the capture replay and is cancelled after 60 seconds. The results are shown in Figure 6, with a layout equivalent to the previous experiment.

Findings: Portest successfully detects the horizontal port scan from the increase in $\mathbb{P}$ when the scan starts. As in the previous experiment, the estimations of $\mathbb{C}$ remain close to their ground truth value. Since there is no vertical port scan, they remain within the same range throughout.

4.5 Server-Side Performance Evaluation

This section evaluates the resource consumption of the server-side software, which is running, for instance, on the SDN controller. Portest aims to offload all resource intensive operations to the TCAM of the switch, which is built to process packets at line rate and with constant overhead. Since no packets need to be processed in software, the server-side resources are only used for the cardinality estimation and

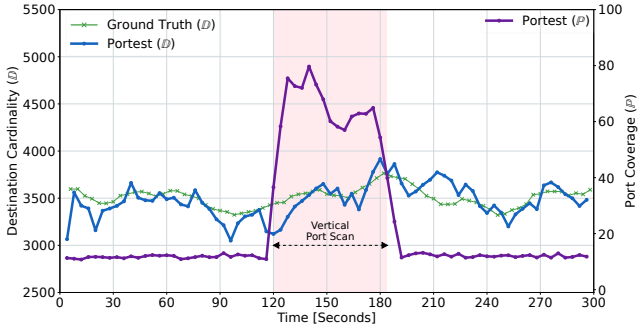

Figure 6: Self-generated vertical port scan using `nmap` with CAIDA ISP traffic capture

attack detection algorithms. During a 15-minute long run, the resource utilization of the processor and memory are monitored. The processor utilization of the Portest process remains below 2% of a single core at all times, and the memory usage is constant at about 100 megabytes. Increasing the traffic load does not affect the resource utilization, since the TCAM entry count and their statistics are always constant.
Findings: The Portest server-side application is lightweight and uses a small amount of resources independent of the traffic load on the switch. No per-packet operations are made, and traffic processing is successfully offloaded to the TCAM.

5 FURTHER ASPECTS

This section discusses some further aspects that are important to the working principles of Portest but have been omitted in the previous explanations for clarity.
Preventing false alarms: A sudden increase in $\mathbb{D}$ could also be caused by an increase in the overall traffic volume (e.g., rerouted traffic after another switch fails), as more traffic likely also contains more different destination IP addresses. Port scans, however, do not have a significant influence on the overall traffic volume. Therefore, Portest also tracks the total traffic volume (sum of all match counters). A detection is only made if $\mathbb{D}$ or $\mathbb{P}$ increases without a simultaneous increase in the total traffic volume.
Saving TCAM capacity: The amount of required TCAM rules can be significantly reduced while retaining equivalent results. In each ruleset (for instance, consider the one shown in Figure 2), the rules with few "don't care" bits are likely never matched for a realistic amount of active devices. By not installing them in the TCAM and assuming their match counter is 0, more than half of the occupied TCAM space is saved while the results stay equivalent [8].
Accuracy of Cardinality Estimations: There are two main factors influencing the accuracy of the cardinality estimations. First, the size of available TCAM, since the cardinality estimations get significantly more accurate when more TCAM entries can be added. More accurate estimations also allow for a reduction of the t parameter since outliers get less likely, enabling the detection of smaller port scans. Second, the uniformity of the individual bits of the destination IP addresses. The estimation is unbiased if all bits are uniformly distributed. In practice, this only applies to the $\approx$ 20 least significant bits of addresses. To prevent estimation bias, the most significant bits have a higher probability of being replaced with "don't care" first during rule generation, and therefore, usually are insignificant to the estimation.
IPv6 and Bit Uniformity: Portest can also be applied to IPv6. However, many of the destination address bits in IPv6 traffic are likely identical for most addresses, which can skew the estimation. Therefore, before installing the rulesets, $2 \cdot 128$ rules can be temporarily added, two for each IPv6 address bit, counting the number of 0 and 1 occurrences. Then, only the bits that are closest to a uniform distribution should be used for the estimation of $\mathbb{D}$, while the others are always set to "don't care" in all rules.

6 CONCLUSION

This paper presented Portest, a novel and scalable approach to detecting security events that processes traffic solely on non-programmable commodity switches. Portest adds rules in the TCAM that probabilistically match packets by destination address and port. By querying the match counters, the destination cardinality and port coverage are estimated to detect horizontal and vertical port scans. The number of required TCAM rules is independent of the traffic amount.

The evaluation using real hardware and realistic traffic datasets shows Portest is able to detect vertical and horizontal self-generated port scans at different scales of background traffic. Additionally, Portest detected a real-world port scan that was inadvertently captured in one of the traffic datasets. Subsequent analysis of the traffic capture at the timestamp of detection confirmed the port scan detection was accurate.

Future work aims to extend and refine the TCAM rules written by Portest to additionally find the source IP address of the port scan. This would allow for an automatic blacklisting of malicious IP addresses. Furthermore, Portest will be evaluated on more modern hardware with a larger TCAM capacity, improving the accuracy especially for smaller port scans. The detection trigger itself could be improved by using machine learning instead of a threshold-based detection.

The main benefit of Portest is the ability to detect port scans with all traffic processed only on *non-programmable* commodity switches, which has not been previously achieved. By using only the match statistics from a constant number of TCAM rules, no additional per-packet overhead is incurred, and network and computational resources for external traffic processing are saved. Portest can also run on programmable switches to save per-packet computational resources by offloading packet processing to the TCAM.

Acknowledgements: This work was supported by the bwNET2.0 project, funded by the Ministry of Science, Research and the Arts Baden-Württemberg (MWK). This work was supported by the Helmholtz Association (HGF) through the Kastel Security Research Labs (POF structure 46.23.01: Methods for Engineering Secure Systems)

REFERENCES

[1] Aristide Tanyi-Jong Akem and Marco Fiore. 2024. Towards Real-Time Intrusion Detection in P4-Programmable 5G User Plane Functions. In *2024 IEEE 32nd ICNP*. IEEE, Charleroi, Belgium, 1–6. https://doi.org/10.1109/ICNP61940.2024.10858543

[2] Abdullah H Alqahtani and John A. Clark. 2022. Enhanced Scanning in SDN Networks and its Detection using Machine Learning. In *2022 (TPS-ISA)*. IEEE, Atlanta, GA, USA, 188–197. https://doi.org/10.1109/TPS-ISA56441.2022.00032

[3] Abdulmohsen Alsaedi, Adel Alshamrani, and Talal Alharbi. 2022. Flow-based Reconnaissance Attacks Detection in SDN-based Environment. *International Journal of Computer Science and Network Security* 22, 9 (Sept. 2022), 747–755. https://doi.org/10.22937/IJCSNS.2022.22.9.98

[4] Dataset. CAIDA. The CAIDA UCSD Anonymized Passive OC48 Internet Traces Dataset. (CAIDA). https://www.caida.org/catalog/datasets/passive_oc48_dataset.

[5] Damu Ding, Marco Savi, Federico Pederzolli, and Domenico Siracusa. 2022. Design and Development of Network Monitoring Strategies in P4-enabled Programmable Switches. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. IEEE, Budapest, Hungary, 1–6. https://doi.org/10.1109/NOMS54207.2022.9789848

[6] Masoud Hasanifard and Behrouz Tork Ladani. 2014. DoS and port scan attack detection in high speed networks. In *2014 11th International ISC Conference on Information Security and Cryptology*. IEEE, Tehran, Iran, 61–66. https://doi.org/10.1109/ISCISC.2014.6994023

[7] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm *(EDBT '13)*. https://doi.org/10.1145/2452376.2452456

[8] Timon Krack and Martina Zitterbart. 2025. Hocarest: Host Cardinality Estimation On-Switch using Randomized Algorithm and TCAM. In *IFIP Networking 2025*. IEEE.

[9] Gordon Fyodor Lyon. 2025. Nmap Network Scanner, Version 7.95. (2025). Software available online at https://nmap.org.

[10] João Monteiro and Bruno Sousa. 2024. eBPF Intrusion Detection System with XDP Offload support. In *2024 IEEE NFV-SDN*. IEEE, Natal, Brazil, 1–6. https://doi.org/10.1109/NFV-SDN61811.2024.10807487

[11] Charles V. Neu, Cassio G. Tatsch, and Roben C. Lunardi; et al. 2018. Lightweight IPS for port scan in OpenFlow SDN networks. In *NOMS 2018*. IEEE. https://doi.org/10.1109/NOMS.2018.8406313

[12] ONF. 2015. OpenFlow Switch Specification, Version 1.5.1. (2015).

[13] Daichi Ono, Luis Guillen, Satoru Izumi, Toru Abe, and Takuo Suganuma. 2021. A proposal of port scan detection method based on Packet-In Messages in OpenFlow networks and its evaluation. *International Journal of Network Management* 31, 6 (Nov. 2021), e2174. https://doi.org/10.1002/nem.2174

[14] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi. 2019. P4 Edge Node Enabling Stateful Traffic Engineering and Cyber Security. *Journal of Optical Communications and Networking* 11, 1 (Jan. 2019), A84. https://doi.org/10.1364/JOCN.11.000A84

[15] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. (Jan 2018). https://doi.org/10.5220/0006639801080116

[16] Sajad Shirali-Shahreza and Yashar Ganjali. 2013. Efficient Implementation of Security Applications in OpenFlow Controller with FleXam. In *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*. IEEE, San Jose, CA, USA, 49–54. https://doi.org/10.1109/HOTI.2013.17

[17] Hua Wu, Ziling Shao, Fuhao Yang, Guang Cheng, Xiaoyan Hu, Jing Ren, and Wei Wang. 2023. PD-CPS: A practical scheme for detecting covert port scans in high-speed networks. *Computer Networks* 231 (July 2023), 109825. https://doi.org/10.1016/j.comnet.2023.109825