Fast Detection of Cyberattacks on the Metaverse through User-plane Inference

Beyza Bütün*†, Aristide Tanyi-Jong Akem*†, Michele Gucciardo*, Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{beyza.butun, aristide.akem, michele.gucciardo, marco.fiore}@imdea.org

***Abstract*—The metaverse is envisioned as a digital world where people can experience an immersive three-dimensional Internet, thanks to the profound integration of different technologies like the Internet of Things (IoT), augmented and virtual reality. From a technical point of view, developing a system of such an unprecedented scale and complexity also opens new challenges in security: a prominent one is the capability to detect and respond to cyberattacks in the shortest time possible, so as not to disrupt the live user experience. In this paper, we discuss how recent advances in user-plane inference can be leveraged to identify malicious traffic generated by IoT devices connected to the metaverse at line rate, ensuring a faster reaction than state-of-the-art approaches where the attack detection is performed in the control plane. We demonstrate the viability of the solution in a programmable network testbed composed of off-the-shelf Intel Tofino switches and with real-world traffic hiding a number of different IoT-based cyberattacks. Our experimental results show that Random Forest models implemented in programmable switches can achieve up to 99% accuracy while using less than 5% of the hardware resources on average in the target case study. Moreover, they quantify the existing trade-off between attack detection precision and user plane resource consumption.**

***Index Terms*—Metaverse, SDN, IoT, in-switch inference, P4**

I. Introduction

The metaverse is envisioned as a digital world where people can live with a digital alter ego, or avatar [1], in what will be the next immersive three-dimensional Internet [2]. Networking technologies are critical enablers for the metaverse, in particular towards guaranteeing the extremely low latency required by many metaverse applications [3].

The fusion of networking technologies required to run the metaverse creates a perfect environment for various security and privacy breaches [4]. As such, the security and privacy of the metaverse users, as well as that of the IoT devices, must be defended from the many threats at multiple points of the network ecosystem. Current state-of-the-art approaches for securing IoT networks rely on SDN paradigms where a controller collects traffic data and monitors the behavior of individual devices [4], so as to profile [5] or isolate [6] malicious connected objects. However, approaches relying on traditional SDN involve unavoidable back-and-forth communication between the user plane that handles the actual data traffic and the control plane where such traffic is analyzed and attacks are detected. This introduces an inherent delay that can range from hundreds of milliseconds to seconds [7] in responding to cyberattacks, which risks to be troublesome for time-sensitive applications in the metaverse [8].

In this paper, we advocate for a novel paradigm where a much faster response to metaverse attacks is realized via user-plane-only inference, and interactions with the control plane are minimized. Our proposal leverage the present proliferation of off-the-shelf programmable data planes like the Intel Tofino ASIC [9], and of domain-specific languages like P4 [10]. Building on these tools, we propose the very first framework for the fast and automated detection of cyberattacks against metaverse IoT devices that relies on pure user-plane inference. By offloading the decision logic from the control plane to the data plane, and perform inference directly in P4 programmable switches, our approach lets us classify traffic for attack detection at line rate, with high throughput and very low latency, hence abiding by the requirements of metaverse applications. With this solution, we make the following contributions.

- We outline a workflow that brings the concept of user-plane machine learning inference into the metaverse ecosystem for line-rate IoT cyberattack detection.
- We introduce a feature, hyperparameter, and model selection process based on an exhaustive grid search, which allows exploring the trade-off between the performance and complexity of models for in-switch operation.
- We implement our solution in a production-grade Intel Tofino programmable switch, using the P4 language, and make our code openly available[1] to the community.
- We conduct an experimental evaluation of the performance of the proposed solution, using a real-world measurement data. Results show how we can detect different attack classes at line rate with an F1 score above 99%, while consuming on average less than 5% of total resources available in the switch.

These contributions let us make a step forward with respect to the state-of-the-art control-plane solutions in terms of detection speed and response time to attacks, and position in-switch attack detection as a viable tool for securing the metaverse.

II. Securing the metaverse at line rate

In the evolution towards a full-fledged three-dimensional Internet, the metaverse will largely rely on cutting-edge communication technologies, and will thus be exposed to many network-related attacks. In order to detect and/or counter such threats, several solutions have been proposed. As a few representative examples, situational awareness has been proposed as a powerful tool for monitoring one or more security domains

[1]Our code is available at https://github.com/nds-group/MetaCom.

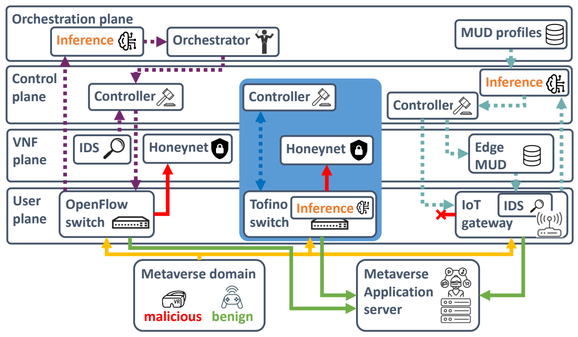


Fig. 1: State-of-the-art solutions overview. Solid lines refer to traffic data: targeted by the inference (yellow), benign (green) and malicious (red). Dashed lines refer to control or feature data flows for the solutions from [6] (purple, on the left) and from [5] (light blue, on the right). Our solution is highlighted (dark blue, in the middle). Figure best viewed in colors.

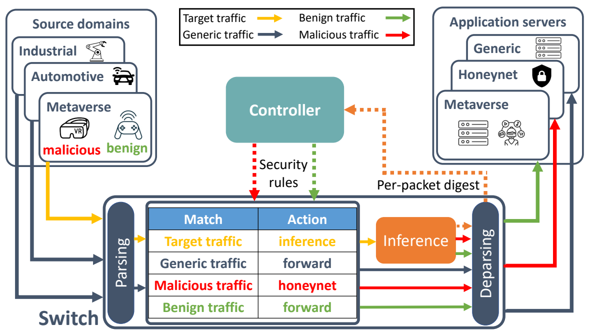


Fig. 2: Overview of the operation of the proposed framework for fast user-plane identification of metaverse cyberattacks.

and reacting promptly [11]; Support Vector Machine (SVM) and clustering has been shown to achieve above 90% accuracy in detecting attacks against an industrial control system simulated via Virtual Reality (VR) [12]; or, Auto Encoders (AEs) have been adopted to learn latent representations of IoT data and show how such representations greatly improve the supervised classification of new attacks [13].

All approaches mentioned above, as well as all other methods in the literature to date, invariably involve deploying the solution in the control and/or orchestration planes. This is illustrated in Figure 1 for two representative proposals introducing sophisticated state-of-the-art architectures based on SDN to identify and isolate malicious behaviors in IoT network [5], [6]. In both workflows, the metaverse domain is defended from attacks via three steps: (i) domain monitoring, (ii) traffic classification, and (iii) malicious traffic isolation. In [6], the monitoring function is fulfilled by a dedicated virtual intrusion detection system (IDS). When the virtual IDS raises an alarm, the controller triggers the switch to send its traffic to an inference block running in the orchestration plane. The classification result is sent to the orchestrator that can select between multiple security policies. The loop is closed by the controller which is responsible for enforcing the chosen policies. The solution described in [5] has a similar workflow, with an IDS running in the CPU of the IoT Gateway triggering the forwarding of flow level metrics to the control plane, where the inference module resides. The classification is performed by feeding a deep neural network with low-level data and MUD profiles taken from databases. After the classification, the controller instructs the switch to drop malicious traffic and updates a MUD module that in turn updates the IDS.

The communications between multiple actors deployed at different planes introduced by approaches such as those above ultimately translates into network overhead and delays that may significantly curb a fast response to attacks. Hence, we propose to reduce the latency in attack detection and response by exploiting the recent advancements in in-switch inference [14]–[20] to perform those tasks directly in the user plane. Figure 1 also illustrates how our framework would perform the separation of benign and malicious traffic directly in the switch, leaving to the controller the only task of modifying the security policies after the attack has been detected. In addition to cutting delays, the approach has the benefit of distributing the cyberattack detection task among many switches, so that the controller is no longer a single point of failure.

A detailed view of the proposed user-plane metaverse attack identification framework is provided in Figure 2. The traffic between the metaverse domain and the corresponding server transits through a switch. The switch has been instructed by the controller to perform classification of traffic in the metaverse domain. During parsing, the information needed to identify the domain and to build the features for the subsequent inference is extracted from the packet. Matching the domain information against a table allows forwarding normally all traffic that is outside the metaverse domain, whereas the features of the metaverse traffic are sent to an inference module implemented within the switch. After the classification, traffic detected as malicious is forwarded to a honeynet for further analysis, and the benign traffic is forwarded to the application server.

III. Practical Implementation

Realizing our proposal for fast user-plane security requires developing suitable machine learning models and embedding them into programmable switches. As model design and training are computationally heavy tasks that go well beyond the switch capabilities, they are performed offline, and only the trained model is deployed in the user plane. Model design and training are carried out through the following three phases.

Feature extraction. We use legacy header fields to compute stateless packet-level features. The features used in our pipeline include TCP flags (ACK, SYN, PUSH, ECE, RESET, FIN), TCP/UDP source and destination ports, packet size, TCP data offset, IP packet length, and Time-to-live (TTL).

Model training. We employ the Scikit-Learn libraries [21] to train Random Forest (RF) models on historical traffic measured in the target network and provided in the form of packet capture (pcap) files. We opt for RF models as they have been repeatedly proven to suit well the architecture of programmable switches [14]–[20].

Feature and hyper-parameter selection. To identify the best combinations of RF model features and hyper-parameters, we run an automated exhaustive grid search. The process explores all RF models with number of trees and tree depth within a predetermined range, and with the features added one by one in the importance order obtained by the Mean Decrease in Impurity (MDI). It picks the smallest feature set that achieves the best performance in the target use case.

Once the final RF model has been identified and trained, we deploy it into the programmable switch. Specifically, we employ the first stage of the Henna model [14] to map the trained RF model onto the Protocol Independent Switch Architecture (PISA) adopted by modern programmable switch ASICs. The mapping is based on that originally proposed by Planter [18], whose code was not available at the time of writing. The process ultimately realizes the inference stage in Figure 2, and comprises the building blocks described next.

Ingress parser. Packets arriving at the switch are processed by the ingress parser, which process the IP and transport-layer protocol headers so as to compose and store in the packet header vector (PHV) the features needed for the inference task.

Ingress control. Based on the extracted features, the associated domain of the packet is determined. In case the packet does not belong to the metaverse domain, it is forwarded according to the standard switching rules. Otherwise, the inference process is triggered by pushing the packet through the match-action (M/A) stages implementing the RF model.

RF mapping. The first set of M/A stages map the RF model to the PISA pipeline. The mapping relies on generating a codeword which embeds the paths to be taken in each tree of the RF model, and is matched against a final set of tables, one per tree, associating each possible codeword (*i.e.*, path) to the classification result Thus, once it has traversed all M/A stages, the packet is tagged with a class by each tree. We refer the reader to [14] and Planter [18] for full details.

Voting system. The final classification decision of the RF model is obtained by a majority vote on the outcome returned by the individual trees. In case of a tie, the certainty value associated to each leaf node, which denotes the accuracy of the classification decision is used: the individual tree output with the highest certainty is adopted.

IV. Performance evaluation

We implement our proposal for in-switch cyberattack detection in an experimental programmable network testbed, as outlined in Section IV-A. We demonstrate the practical viability of the solution on publicly available real-world network traffic measurements of IoT-based threats presented in Section IV-B, using performance metrics in Section IV-C.

A. Hardware setup

Our testbed consists of three Edgecore Wedge100 programmable switches with Intel Tofino BFN-T10-032Q chipsets and 32 100GbE QSFP28 ports, and two servers equipped with Intel 8-core Xeon processors at 2GHz, 48GB of RAM, and QSFP28 interfaces. The testbed is thus fully 100Gbps-capable.

TABLE I: Performance of different RF models in detecting cyberattacks in the IoT/IIoT space, in terms of weighted F1 score, true positive rate (TPR), false positive rate (FPR), true negative rate (TNR) and false negative rate (FNR).

	Weighted F1-Score		**TPR**		**FPR**		**TNR**		**FNR**	
Model	Python	Switch	Python	Switch	Python	Switch	Python	Switch	Python	Switch
RF(5,3,7)	98.34	98.33	99.66	99.67	2.84	2.91	97.16	97.09	0.34	0.33
RF(7,3,7)	98.53	98.89	98.35	98.96	1.11	1.20	98.89	98.80	1.65	1.04
RF(9,3,10)	98.92	98.85	99.89	99.77	2.72	2.68	97.28	97.32	0.11	0.23
RF(10,3,9)	99.28	99.27	99.93	99.93	0.74	0.77	99.26	99.23	0.07	0.07

The switches run an Open Network Linux (ONL) operating system. To automatically conduct the initial configuration of the switch and perform tasks such as setting up switch ports and loading the mapped RF model table entries, we use a Python controller program that interacts with the Barefoot Runtime Interface (BRI). Then, by replaying pcap traces using Tcpreplay [22], we employ 100Gbps connections to inject traffic into the switch from one server to the other.

B. Cyberattack dataset

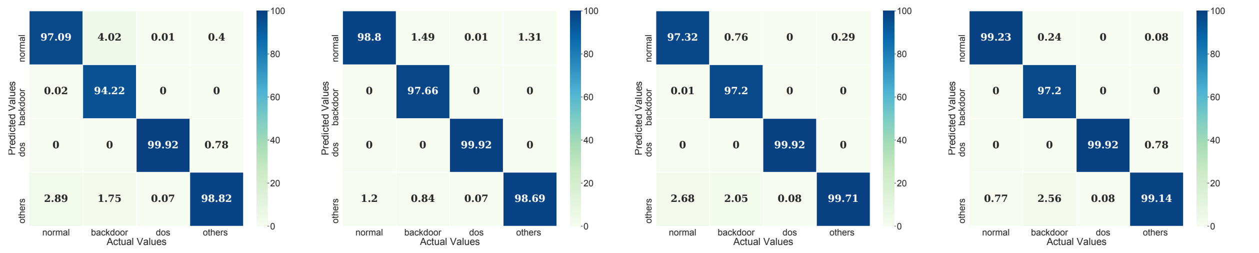
To demonstrate how in-switch machine learning inference can detect and classify attacks in metaverse IoT networks at line rate, we use the ToN-IoT dataset [23] captured from a representative medium-scale testbed deployed using several virtual machines to manage the interconnections between the network domains of Cloud, Fog, and Edge. The testbed includes IoT, industrial IoT, and non-IoT devices in the Edge network domain. The dataset is captured using parallel processing to collect benign and several cyberattack events [23], such as Scanning, Denial of Service (DoS), Distributed Denial of Service (DDoS), Ransomware, Backdoor, Injection, Cross-site Scripting (XSS), Password, Man-In-The-Middle (MITM).

C. Detection goal and metrics

Our inference objective is to classify packets transiting in the switch by tagging them as either *benign* or belonging to one of three attack classes: *DoS*, *Backdoor*, and *others*. The last class includes Scanning, DDoS, Ransomware, Injection, XSS, Password Cracking, and MITM attacks. The cyberattack detection performance is evaluated using metrics derived from four key measures of a classification task, *i.e.*, true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), which are True positive rate (TPR) = TP/(TP+FN), False positive rate (FPR) = FP/(FP+TN), True negative rate (TNR) = TN/(TN+FP), False negative rate (FNR) = FN/(FN+TP), and F1 score = 2TP/(2TP+FP+FN). For the classification quality, we utilize the *weighted* F1 score that is the mean computed for each class weighted using the number of samples of each class in the data. We report all these values as percentages for the sake of interpretability.

V. Results and discussion

Based on the offline feature and hyper-parameter selection phase we select a choice of four RF models with varying complexity and yielding different inference accuracy. We refer to these models as $RF(5,3,7)$, $RF(7,3,7)$, $RF(9,3,10)$, and $RF(10,3,9)$, respectively, where the parameters indicate the depth, trees and number of features used by each model.

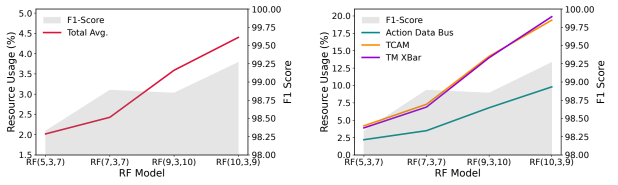


(a) RF(5,3,7) (b) RF(7,3,7) (c) RF(9,3,10) (d) RF(10,3,9)

Fig. 3: Confusion matrices of the four selected RF models implemented in an Intel Tofino switch, computed based on the cyberattack detection results recorded in the experimental programmable network testbed.

TABLE II: Resource utilization by the cyberattack detection module as a percentage of the total resource available.

Resource	Models			
	RF(5,3,7)	RF(7,3,7)	RF(9,3,10)	RF(10,3,9)
Action Data Bus Bytes	2.20	3.50	6.80	9.80
Logical Table ID	8.90	8.90	10.40	9.90
SRAM	1.60	1.60	2.00	2.20
TCAM	4.20	7.30	14.20	19.40
Ternary Match Input Crossbar	3.90	6.90	14.00	19.90
VLIW Instruction	4.40	3.90	4.40	4.40
Total Avg. Resource Usage	2.02	2.43	3.59	4.40
Match & Action Stages	7	7	9	9



(a) (b)

Fig. 4: Accuracy and resource usage of the in-switch models. (a) All resources. (b) Action Data Bus, TCAM, and TM Crossbar.

A. Classification accuracy

The classification performance of our solution in identifying attacks is summarized in Table I in terms of weighted F1 score, TPR, FPR, TNR, and FNR. For each metric, we report results obtained with the python implementation of the RF models, which runs on a regular CPU, and with the P4 implementation in the programmable switches of our experimental testbed. This allows us to comment on if and to what extent the constraints of in-switch inference affect the performance of the machine learning solution when it is run directly in the user plane, compared to a counterpart operating in the control plane as commended by state-of-the-art security solutions for the metaverse based on SDN.

Overall, we correctly identify attacks with a weighted F1 scores in the range 98.34% to 99.28% depending on the RF model. TPR values are well above 99% in three of the four models; in the case of the most complex $RF(10,3,9)$ model, the switch correctly detects more than 99.9% of all malicious packets. TNR values are slightly lower for the less complex RF models, indicating that benign traffic is slightly more difficult to classify correctly, most likely due to its more heterogeneous nature; yet, the value stays above 97% in the worst case. The FPR is always less than 3%, and the FNR is never above 0.5%, showing the low error rate of the in-switch solution.

When comparing the implementation programmed in P4 and running in the switch with that CPU-based in python, differences are overall negligible. These discrepancies are ascribed to the fact that there is a small fraction of packets (less than 0.4%) for which a tie occurs in the majority vote of the RF trees, and all trees yield the same certainty values. The python implementation embeds further tie-breaker mechanisms to deal with these situations, which are however too complex to implement in the switch, which is forced to take random choices. Yet, overall, the performance of a user-plane implementation is well aligned with those of legacy control-plane solutions, proving the viability of our approach.

A detailed breakdown of the performance of the user-plane solution is presented in Figure 3, where the complete confusion matrices are reported for the four RF models. Benign normal traffic is correctly classified in over 97% of cases with the simplest $RF(5,3,7)$ model, and in more than 99% of packets with the most complex $RF(10,3,9)$ model. Backdoor attacks are detected above 97% of the time across all the models, except for the simplest $RF(5,3,7)$ model where performance drops to 94%. DoS attacks are detected in over 99% of situations with all the models, while the other attacks are classified with a score of over 98% with the simpler models and over 99% with the other two models. These figures let us conclude that user-plane cyberattack detection not only separates malicious traffic from normal traffic, but can further tell apart three types of cyberattacks with high accuracy.

Finally, it is worth noting that all in-switch results above are obtained by processing packets at the same rate at which the traffic data goes through the switching hardware. This implies that the cyberattack detection occurs transparently as packets traverse the switching fabric, with a negligible added latency to regular user-plane operation. Indeed, we measured an average inference latency in the range of 73–91 ns per packet, which proves how a pure user-plane implementation of a security mechanism for the metaverse yields huge latency improvement over traditional control-plane approaches that involve communication delays of milliseconds to seconds [7].

B. Resource usage

As programmable switches are highly constrained in terms of available resources, it is important to assess the resource footprint of the cyberattack detection functionality once it is deployed in the hardware. To gather statistics on resource usage, we use the Intel P4 Insight tool, which delivers a thorough analysis of the compiled P4 programs in the target Intel Tofino ASICs, and of their mapping to specific hardware resources. The results are recapitulated in Table II.

On average, the RF models we use consume less than 5% of the total available resources on the hardware. When juxtaposed to the standard P4 program for core L2/L3 switching, *i.e.*, `switch.p4`, integrating a cyberattack detection functionality grows resource usage by around 10% over what the baseline program already needs. Usage is however not uniform across different types of resources: in most cases, requirements are fairly limited, between 0% and 10%, but exceptions exist for the ternary content-addressable memory (TCAM) and ternary match input crossbar (TM Crossbar) that are used for matching codewords that include wildcards. In these cases, the RF models could consume 4% to 20% of the switch resources. In all cases, consumption is low enough for all types of resources and shows that in-switch machine learning inference is a potentially viable solution for detecting attacks against IoT devices in the metaverse, while still allowing coexistence with other legacy switching functions.

Importantly, different RF models have fairly diverse needs in terms of switch resources, especially for those most intensively used for inference, like the aforementioned TCAM and TM Crossbar, or the Action Data Bus Bytes. By cross-correlating these results with the performance in Table I, we can quantify the trade-off between accuracy and requirements of the cyber-attack detection functionality implemented in a programmable switch. Figure 4 summarizes the trade-off, by comparing how models of increasing complexity consume resources and affect the weighted F1 score, which we consider here as an overall measure of inference quality. Figure 4a shows that the total resource utilization grows in a way that is fairly proportional to the detection accuracy. Figure 4b confirms that this is the case also for those individual resources that are affected the most by the model hyper-parameters.

Note that the scales are different: *e.g.*, increasing the maximum tree depth from 5 to 10 leads to a 1% gain in F1 score and a 2% drop in FPR, but results in a 15% and 16% absolute increase in the amount of required TCAM and TM Crossbar. This creates a space for the operator to decide whether to deploy a more complex user-plane cyberattack identification and attain near-perfect accuracy, or spare resources and accept a 3% error in immediately detecting attacks to the metaverse.

VI. Conclusions

We propose a machine learning solution for detecting cyberattacks against IoT devices in the metaverse, which is fully deployed in the user plane. Our design operates at line rate, and removes the communication latency with the control plane that affects current SDN-based approaches for securing the metaverse. Experiments with off-the-shelf hardware and real-world traffic show that we can detect cyberattacks with 99% accuracy while consuming on average less than 5% of the total hardware resources, and with latency well below 100 ns.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 101017109 "DAEMON" and the Marie Skłodowska-Curie grant agreement no. 860239 "BANYAN", from the CHIST-ERA grant no. CHIST-ERA-20-SICT-001 "ECOMOME", and from the Spanish Ministry of Economic Affairs and Digital Transformation and the European Union-NextGenerationEU through the UNICO 5G I+D project no.TSI-063000-2021-52 "AEON-ZERO". We thank the Intel Connectivity Research Program for their support.

References

[1] J. Xu et al., "Metaverse: The vision for the future," in *CHI EA '22*. NY, USA: ACM, 2022.
[2] L-H. Lee et al., "What is the metaverse? an immersive cyberspace and open challenges," *arXiv*, 2022.
[3] Y. Fu, C. Li, F. R. Yu, T. H. Luan, P. Zhao, and S. Liu, "A survey of blockchain and intelligent networking for the metaverse," *IEEE Internet of Things Journal*, 2022.
[4] Y. Wang et al., "A survey on metaverse: Fundamentals, security, and privacy," *IEEE Communications Surveys & Tutorials*, 2022.
[5] P. Krishnan et al., "MUD-based behavioral profiling security framework for software-defined IoT networks," *IEEE Internet of Things Journal*, vol. 9, no. 9, 2022.
[6] A.M. Zarca et al., "Virtual IoT honeynets to mitigate cyberattacks in SDN/NFV-enabled IoT networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, 2020.
[7] K. He et al., "Measuring control plane latency in SDN-Enabled switches," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, New York, NY, USA, 2015.
[8] J. Yu et al., "6G mobile-edge empowered metaverse: Requirements, technologies, challenges and research directions," *arXiv*, 2022.
[9] Intel, "Tofino Programmable Ethernet Switch ASIC," 2016.
[10] P. Bosshart et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, 2014.
[11] J. Woodward et al., "Analytic review of using augmented reality for situational awareness," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2022.
[12] Z. Lv et al. , "Industrial security solution for virtual reality," *IEEE Internet of Things Journal*, vol. 8, no. 8, 2021.
[13] L. Vu et al., "Learning latent representation for IoT anomaly detection," *IEEE Transactions on Cybernetics*, vol. 52, no. 5, 2022.
[14] A.T-J. Akem et al., "Henna: Hierarchical machine learning inference in programmable switches," in *NativeNi '22*. NY, USA: ACM, 2022.
[15] C. Zheng et al., "IIsy: Practical in-network classification," *arXiv*, 2022.
[16] G. Xie et al., "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE INFOCOM 2022*.
[17] B-G. Coralie et al., "pForest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.
[18] C. Zheng et al., "Automating in-network machine learning," *arXiv*, 2022.
[19] B.M. Xavier et al., "Programmable switches for in-networking classification," in *IEEE INFOCOM 2021*, 2021.
[20] X. Zhang et al. , "pHeavy: Predicting heavy flows in the programmable data plane," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, 2021.
[21] "Scikit-learn: Machine Learning in Python, author=F. Pedregosa et al." *Journal of Machine Learning Research*, vol. 12, 2011.
[22] A. Turner et al., "Tcpreplay," 2013. [Online]. Available: https://tcpreplay.appneta.com/
[23] A. Alsaedi et al., "TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems," *IEEE Access*, vol. 8, 2020.