# E-Commerce Bot Traffic: In-Network Impact, Detection, and Mitigation

Masoud Hemmatpour[+]
*Computer Science Department*
*Arctic University of Norway (UiT)*

Changgang Zheng
*Department of Engineering Science*
*University of Oxford*

Noa Zilberman
*Department of Engineering Science*
*University of Oxford*

***Abstract*—In-network caching expedites data retrieval by storing frequently accessed data items within programmable data planes, thereby reducing data access latency. In this paper we explore a vulnerability of in-network caching to bots' traffic, showing it can significantly degrade performance. As bots constitute up to 70% of traffic on e-commerce platforms like Amazon, this is a critical problem. To mitigate the effect of bots' traffic we introduce In-network Caching Shelter (INCS), an in-network machine learning solution implemented on NVIDIA BlueField-2 DPU. Our evaluation shows that INCS can detect malicious bot traffic patterns with accuracy up to 94.72%. Furthermore, INCS takes smart actions to mitigate the effects of bot activity.**

***Index Terms*—component, formatting, style, styling, insert**

## I. Introduction

The rapid growth of e-commerce has led to an increased prevalence of automated bots engaging in fraudulent activities [1], [2]. Bot is a software application programmed to execute specific tasks to simulate human in social or web activities [3]. Despite online businesses' continuous efforts to enhance user experience through the adoption of modern technologies [4], [5], the disruptive presence of malicious bots poses a significant threat to the revenue and overall success of companies [6].

An instance of modern technologies that hold the potential to bring a transformative shift for backend e-commerce systems includes smart network devices [7], [8]. These network devices can perform customized computations (see Section II-C). This new paradigm of computation known as In-network computing or NetCompute, refers to the execution of programs typically running on end-hosts within network devices [9]. Such network devices can be programmed through traditional programming languages such as hardware description language (HDL) and C/C++ languages. However, a specialized language, Programming Protocol Independent Packet Processors (P4), was developed for configuring how network devices process and forward packets within the network devices [10]. This approach of computation is gaining importance in various fields as it offers advantages like reduced latency, improved scalability, and enhanced efficiency for networked applications [7], [8]. A successful class of applications of in-network computing is in-network caching where network devices quickly reply to a user request with cached values in the network device. However, exploiting in-network caching in the backend of e-commerce system can bring security risks which need to be discovered to avoid any vulnerability.

In this paper, we review the characteristics of bot traffic in real-world e-commerce websites and analyze how these characteristics can bring vulnerability while using in-network caching in the backend. We outline how the presence of bot traffic can lead to higher cache misses of legitimate users. Furthermore, we formulate experiments to investigate and demonstrate such influence on the efficiency of in-network caching.

Studies have demonstrated that even a marginal improvement in the hit rate of just 1% of cached data can result in a substantial reduction of over 35% in application layer latency [11]. Therefore, we offer an on-the-fly P4 based in-network machine learning (ML) solution to detect such malicious bot traffic and mitigate its impact. We implement our solution in the Data Processing Unit (DPU), which, to the best of our knowledge, is the first P4 based machine learning solution in the DPU. Our main contributions in this research can be summarized as follows:

- Investigating a vulnerability of in-network caching, and providing a dataset based on the characteristics of bot traffic in real-world e-commerce websites,
- Exploring the impact of the investigated vulnerability on the latency of normal users,
- Proposing an in-network solution to detect bot traffic and mitigate its impact using machine learning.

In section II, we present an overview of bot traffic in e-commerce, aiming to provide insights into current strategies, challenges, and advancements in addressing this critical issue. Furthermore, we describe in-network computing and the relevant requirements for our study. Section III describes the in-network caching vulnerability to malicious bot traffic. Section IV presents the characteristics of malicious bot in real-world e-commerce traffic and outlines the methodology used to generate a dataset based on these characteristics. Section V describes the system design of our solution for this problem. In Section VI, we present the evaluation of our analysis by designing different experiments. Then, we illustrate how a P4 based machine learning solution can successfully detect malicious bot traffic with high accuracy. In Section VII, we

[+]Additionally, the author has an affiliation with Simula Research Laboratory (SRL). The research project was initiated at SRL and subsequently continued at UiT.

delve into a discussion of our proposed solution and directions for future work. Finally, in Section VIII we draw conclusions.

## II. Background

In this section, we provide a foundational background for our research, encompassing aspects such as e-commerce bot traffic, strategies for bot detection, and the concept of in-network computing.

### A. E-commerce bot traffic

By June 2019, there were more than 4.5 billion Internet users worldwide [12]. Nowadays, Internet users are not limited to human and technologies such as Internet-of-Things (IoT) even increase up users [13]. A significant part of the overall Internet traffic is generated by bots [2], [14]. Legitimate services, such as search engines, use bots to crawl websites and power their products. However, according to reports, more than half of bot traffics are potentially harmful [15]. Malicious bots can cause several issues, such as Distributed Denial of Service (DDoS) attacks [2], generating fake news [16], web scrapings [1], or shilling attacks in recommender systems [17]. Among industries, e-commerce sees a wide range of bot attacks with $\approx 20\%$ malicious bot traffic [18], [19]. E-commerce ranks fifth in the intensity of bad bot traffic and first in the volume of sophisticated bot traffic [20]. Bot traffic is a long-standing problem for companies such as Amazon [21], causing a huge economic impact. There are different types of bots which can cause problems for e-commerce such as manipulating product ranks [17] and increasing data access latency [21]. In more detail, malicious bots generate content data, such as user ratings and reviews to manipulate recommendation rankings [17]. Considering the fact that recommender systems drive 35% of online sales, manipulating product reviews or ratings will have a big impact on a company's revenue. Furthermore, latency is an important factor in e-commerce businesses such that AOL reports that the average page views can drop from 8 to 3 in the slower response time, and Amazon reports that by adding 100 ms, sales drop by 1% [21], [22].

### B. E-commerce bot detection

Many bots try to emulate navigational behaviors of legitimate users at a very high sophistication level [15]. This creates several challenges for companies with online businesses. Therefore, bot detection is an important part of an online business. E-commerce bot detection involves the identification and mitigation of automated bots that interact with online e-commerce platforms. Bot technology continues to evolve which makes bot detection even harder. These bots can have various purposes [21] and different approaches have been proposed to detect bots such as sitemap-based [3], log analysis [23], behavioral analysis [14], and machine learning techniques [24], [25].

A large body of research has been dedicated to the problem of bot traffic analysis, characterization, and classification [15], [23]. However, bot detection has been mostly addressed in the offline scenario, working on historical session data and very few studies have dealt with the problem of identifying bots on-the-fly [2]. The ability to identify bots in real time is of crucial importance for the security and performance of an e-commerce website since it makes it possible to mitigate threats before the end of bot visits and, thus, to limit the impact of their presence. Because the amount of business of e-commerce websites primarily depends on search engines for product discovery [26], in this paper vulnerability that malicious bot can cause on search engines while in-network caching is exploited in the backend of website is investigated.

### C. In-network computing

Traditional data networks passively transport data from one node to another and the role of computation within such networks is limited. Now, Programmable Network devices (PNDs) such as SmartNICs, and programmable switches allow to run customized computation on the network devices themselves. In the following, we delve into two significant facets of in-network technologies.

- **In-network caching** refers to a technique where caching mechanisms are implemented within the network infrastructure itself. In more detail, a query statistic monitors the requests in the traffic and stores the frequent ones. The primary purpose of in-network caching is to improve the efficiency and performance of content delivery, reduce latency, and alleviate the load on upstream servers. Caching can be implemented in the SmartNIC such as *Lake* [8] or in Smart Switch such as*NetCache* [7].
- **In-network machine learning** refers to the offloading and integration of the machine learning inference process directly within PNDs. Specifically, this approach conducts the offline training of machine learning models, followed by the mapping of trained machine learning models into PNDs for online inference [27]. In-network machine learning leverages the computational capabilities of network devices to perform various machine learning tasks, thereby enhancing the efficiency, adaptability, and intelligence of the network itself. To date, in-network machine learning has demonstrated support for a range of models, for example, Decision Trees (DT), Random Forests (RF), $k$-Means, $k$-Nearest Neighbors (KNN), Neural Networks (NN), and Naïve Bayes.
  Leveraging prototyping frameworks like Planter [28] and DINC [29], in-network machine learning algorithms have conducted applications across various domains, notably in tasks such as anomaly detection [28]–[30], traffic classification [31], load balancing [32], and financial market prediction [33]. Nevertheless, while some references present in-network machine learning for attack detection (especially for DDOS) [30], [34], the case of attacks on caching systems is unique, because none of the preceding works have explicitly addressed its integration within caching systems for its attacks.

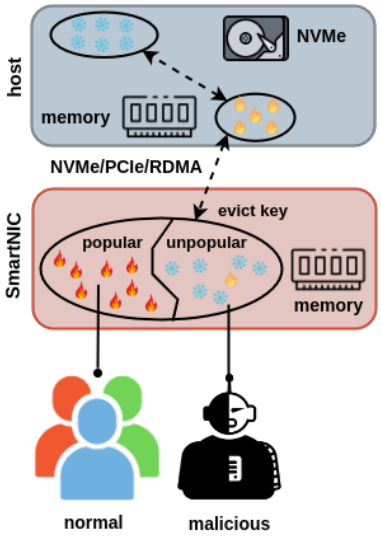



Fig. 1: Vulnerability of caching.

## III. Caching Vulnerability

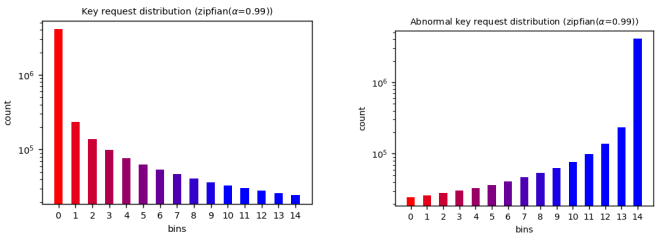
In e-commerce and many other Internet services rendering even a single web page often requires hundreds or thousands of storage accesses [35]. So, as these services scale to billions of users, system operators increasingly rely on caching to meet the necessary throughput and latency demands [7], [36]. Caching has been exploited in many different social community and e-commerce websites such as Wikipedia [37], Amazon [38], Alibaba [39], and Facebook [40]. Search engines in e-commerce websites particularly exploit caching to improve scalability, performance and user experience [41]. For example, Amazon developed *ROSE*, a caching system, to improve its search quality for different business metrics [42]. Despite the benefits of caching, especially within an in-network context, this technology face several challenges that can impact its effectiveness and adoption. Mainly, in-network caching presents two fundamental challenges, namely, the identification of dynamically shifting hot items and the management of memory constraints within PNDs. In this research, we delve into these challenges, recognizing their potential to render in-network caching vulnerable.

As depicted in Figure 1, regular users typically request popular items, leading to the storage of these items within the in-network caching system. However, malicious bots follow a different pattern, actively soliciting less popular items and attempting to inject counterfeit popular items into the caching system. Due to the constrained memory capacity of PNDs, the caching system initiates eviction procedures, replacing legitimate popular items with the newly introduced false items by the bots [43]. This eviction process results in a noteworthy latency increase for normal users. Furthermore, due to the dynamic nature of hot items requested by normal users, bots may persistently prioritize keeping items in the cache that were previously hot but may no longer hold the same level of popularity. In Section VI, we have devised specific scenarios to quantify and assess the extent of this performance impact.

## IV. In-network caching attack

Gaining access to data for research in e-commerce and related areas like search and recommendation has proven to be a challenging endeavor [26]. We conducted an analysis on several pre-existing datasets such as Twibot-20 [44], botnet [45], Wikimedia API [46], Amazon [47], and Alibaba benchmark generator [48]. However, these datasets did not meet our criteria due to either lacking sufficient annotations or lacking essential query and user information. As a result, we conducted a thorough review, which encompassed an analysis of the characteristics of the web requests [49], web traffic [50], workload [51], bot traffic [2], and network caches [52]. Then, we formulated a set of guiding principles for the precise generation of a bot traffic dataset. The following are the key principles we adhered to during the dataset creation process:

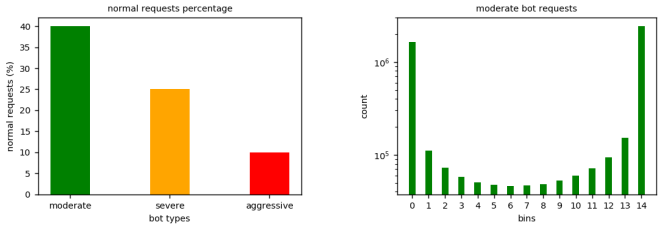
- Search queries made by bots are often long search queries and tend to search for not so popular items in the e-commerce search engine [2]. Therefore, search queries of bots are often responded with much fewer items, more exactly, about an order of magnitude fewer than the items returned for a search query made by normal users.
- User requests follow a Zipfian distribution [49]. This observation signifies that specific segments of the data experience a higher volume of requests, a pattern that evolves over time. To mimic user behavior accurately, we proceeded to generate 5 million requests using a Zipfian distribution with a $\alpha$= 0.9. Figure 2 illustrates the distributions of keys for normal and bots. As can be seen in 2a, there are more request for some part of data (i.e., popular/hot items) and less request for some other (i.e., unpopular/cold items). However, as Figure 2b shows, cold items for normal users consider as hot items for bots [2].



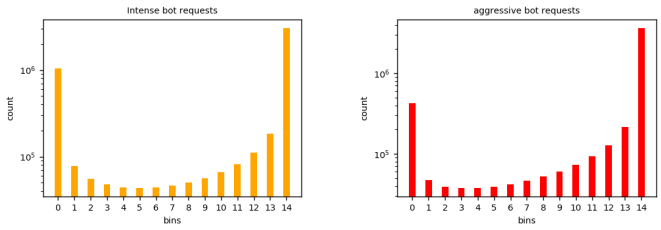

(a) Normal requests distribution (b) Bot requests distribution

Fig. 2: Red/Blue colors highlight popular/unpopular requests of normal user.

Our analysis affirms that bots do not continually request popular items, and they may occasionally submit regular requests as a means to conceal their activities. As a result, we have classified bots into three distinct categories within our dataset: (1) *Moderate*, (2) *Intense*, and (3) *Aggressive*. As Figure 3 illustrates, *Moderate* bots make requests for popular items 40% of the time and for unpopular items 60% of the time. *Intense* bots, on the other hand, request unpopular items 75% of the time and popular items 25% of the time. *Aggressive* bots predominantly request unpopular




(a) Popular bot requests. (b) Moderate bot requests.




(c) Intense bot requests. (d) Aggressive bot requests.

Fig. 3: Types of bots in dataset.

items, with a distribution of 90% for unpopular items and 10% for popular items.

- The average time interval between consecutive requests made by bots is five times shorter than normal users [2]. Consequently, we incorporate this factor into our request generation process, producing two distinct sets of time intervals for normal and bot traffic. As depicted in Figure 4, normal users follow a Gaussian distribution with parameters $\mu$=70 and $\sigma$=4, while bot requests are characterized by parameters $\mu$=14 and $\sigma$=1. $\mu$ represents the mean or average value of the distribution, and $\sigma$ is standard deviation which shows the dispersion of the data points in the distribution.

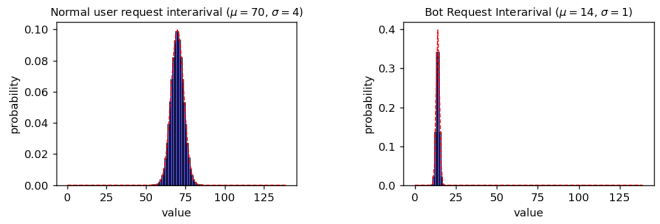



Fig. 4: Time interval between requests.

We have made both the dataset and the experiments publicly available to facilitate reproducibility[1].

When a bot or bots issue requests for an unpopular item, they have the potential to deceive the system into perceiving this item as popular. Given the inherent constraints of memory within SmartNICs, there exists a significant likelihood that caching system may need to evict one of the currently stored item to accommodate the introduction of this new item into the cache. Consequently, when a regular user requests the evicted item, caching system must once again evict another item from the cache to make room for the popular item. As a result, a typical user incurs the following penalty for a cache miss:

$$cache\ miss\ cost = load\ data + add\ data$$

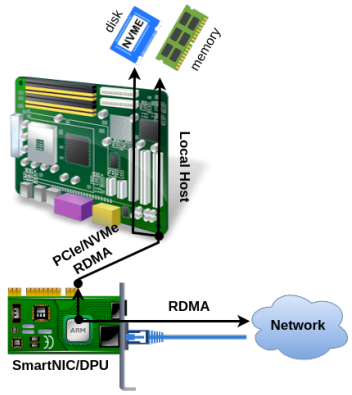



Fig. 5: load data from different locations.

To begin with, the process involves reloading the popular item that was previously evicted from the cache. As depicted in Figure 5, this data may either reside within the local host [8], [53] or be situated on a remote host [54]. The latency associated with data retrieval hinges upon the specific SmartNIC in use and its supported interfaces.

Following the data retrieval from its respective location, the next step involves adding it back to caching system. Adding to the caching system can increase the latency of other normal user data access. To assess the effect on the latency experienced by a normal user, we established a testbed with the NVIDIA BlueField-2 DPU. This DPU is equipped with 8 Armv A72 cores, 16GB of on-board DDR4 memory, a PCIe Gen 4.0 interface, and InfiniBand dual ports of EDR/HDR 100. Furthermore, we installed Memcached version 1.5.22 and libmemcached version 1.0.18 as part of our experimental setup.

Furthermore, we designed an experiment in which a client is reading data (keys and value size=128 Bytes) while an intruder interferes by three different operations: 1) reading data, 2) adding data, and 3) adding data which causes an eviction, occurring when the amount of available memory for caching system runs out. The intruder performs its operations with a delay. As Figure 6 shows, larger introduced delays between operations by an intruder will reduce the interference. This, in turn, can lead to a decrease in the latency of data access experienced by the client. By spacing out, the intruder is essentially allowing more breathing room for the client operations to proceed without disruption, resulting in improved response times for the client. As can be seen, when read and add operations of intruder overlap with the client, the resultant impact on the client performance is nearly identical. However, we limited the memory of Memcached to 512MB and noticed that if the intruder adds a key which causes an eviction then the latency of data access for the client significantly increased

[1] https://github.com/mashemat/inet_bot_traffic_detection

(≈ 2x) since it requires to reload and add it to the Memcached. Size of available memory has a big impact on the performance of the caching system [55]. Given the constrained memory capacity, particularly in PNDs, the likelihood of needing to add new items to the cache, coupled with the eviction of currently stored items, is significantly elevated.

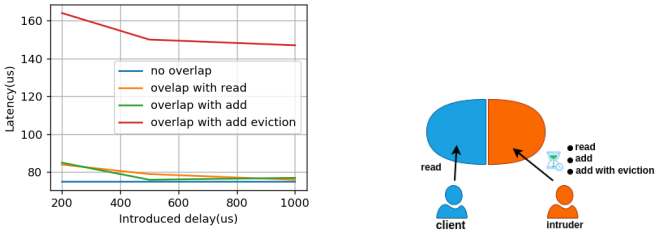



Fig. 6: Data access latency with an intruder.

## V. System Design of INCS

In this work, we propose INCS, a mitigation solution for attacks on in-network caching. INCS realizes an on-the-fly in-network bot detection mechanism using in-network machine learning inference.

The in-network caching system, shown in Figure 7, has a controller that is responsible to assess the popularity of a requested item. If the item meets a certain popularity threshold, it is stored on the PND within a caching system, such as Memcached. To safeguard the caching system, a machine learning algorithm is used to classify incoming requests, detecting and acting on bot-generated queries.

INCS runs on a programmable network target, such as a DPU using Behavioral Model Version 2 (BMV2) [56] or Open vSwitch (OvS). The in-network machine learning algorithms are realized using Planter [28], observing all incoming requests before the controller. INCS provides feature extraction required for the machine learning, as well as standard switch functionality, and the two are executed in parallel.

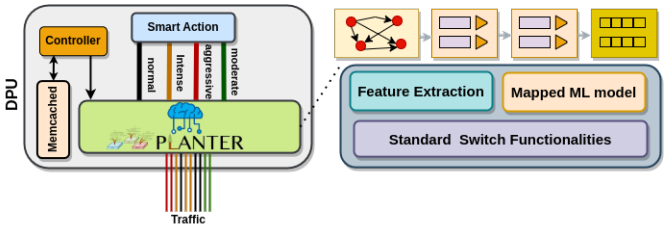



Fig. 7: Architecture of INCS bot detection.

INCS extracts features from incoming traffic, and performs the filtering action and bot traffic classification in real time. It takes into account two pivotal features of bot traffic for its classification: the time intervals between consecutive requests and the specific items for which these requests are made [2], with a focus on identifying unpopular items. The feature extraction in the M/A pipeline computes the time interval of requests for each flow and the amount of items that each flow requests. These requests will subsequently be passed to the machine learning workflow.

The machine learning code in the data plane is generated using Planter. The system provides data and configurations to Planter, train the model on the server, and maps the trained model to the programmable pipeline. In the data plane, the workflow takes the extracted features and determines the popularity levels of the requested item [7]. Afterwards, it classifies the traffic as *Aggressive*, *Intense*, *Moderate*, and *Normal*.

Each class of traffic is assigned a different action. *Aggressive* bot traffic can be immediately dropped. The bandwidth of *Intense* traffic can be reduced. Sometimes, *Moderate* traffic can be further processed. However, if multiple *Moderate* traffic flows request the same unpopular items, it indicates that a set of bots are striving to introduce a set of unpopular items as a popular item. In such a case, all the *Moderate* traffic flows requesting the same items can be dropped.

## VI. Evaluation

To assess the effectiveness of INCS, we deployed INCS on BlueField-2 DPU's ARM cores using several trained machine learning models. The generated datasets (Section 4) are used for the attack. Table I shows the accuracy of bots detection across various models.

| Models | Moderate | Intense | Aggressive | Mix |
|---|---|---|---|---|
| DT | 80.28 | 87.36 | 94.72 | 78.94 |
| RF | 80.42 | 87.36 | 94.72 | 78.94 |
| Naïve Bayes | 78.64 | 85.54 | 93.34 | 78.80 |
| KNN | 56.56 | 85.86 | 93.20 | 76.00 |
| K-means | 77.06 | 83.94 | 90.14 | 76.38 |

TABLE I: Accuracy of bot detection by different models.

As the bot behaves more aggressively, it is easier to detect it. For instance, as the table shows, Random Forest (RF) achieves a bot detection accuracy of 80.42% for *Moderate* bot activity, 87.36% for *Intense* bot activity, and 94.72% for *Aggressive* bot activity.

False Positive Rate (FPR) and True Positive Rate (TPR) are important metrics to provide insights into a model's performance in terms of correctly identifying negatives and positives, respectively. False Positive (FP) occurs when a model incorrectly identifies a bot-generated query as normal, and a False Negative (FN) occurs when a model incorrectly identifies a normal query as a bot. True Positive (TP) refers to a correct classification of a normal query, and a True Negative (TN) occurs when a bot query is detected. $FPR = FP/(FP+TN)$ and $TPR = TP(TP + FN)$. Table II shows the (FPR,TPR) of the RF model.

| Moderate | Intense | Aggressive | Mix |
|---|---|---|---|
| (0.38,0.99) | (0.25,1) | (0.1,1) | (0.28,1) |

TABLE II: (FPR,TPR) of RF model. Positive indicates normal traffic.

INCS using Random Forest has high TPR for all types of bots attacks, indicating that the model is effective at identifying

normal traffic when it is present. In other words, it doesn't mistakenly predict normal traffic as bot. This is an important factor since it can have an impact on the user experience of regular users. As FPR here indicates bot queries mistakenly considered normal traffic, it can be seen that INCS using Random Forest correctly identifies 72% of bot traffic in a mixed environment, and performs even better with intense and aggressive attacks.

## VII. Discussion

Programmable network devices provide powerful enough processing capabilities on network devices to implement security programs on it [57]. However, analyzing any network traffic raises important privacy and ethical concerns. Implementing privacy-preserving techniques and adhering to data protection is a priority. In our solution, we have taken into account these considerations by designing it in such a way that it does not depend on any sensitive or protected data, and users can maintain their anonymity. Moreover, privacy is not the primary focus of this study and our INCS system makes a contribution to safeguarding in-network caching from malicious bot traffic. It possesses the capability to detect various types of bots and implement diverse actions to mitigate the effects caused by these bots.

Identifying and mitigating the effects of bot traffic reduce the number of less frequently accessed items within the caching system. This, in turn, results in memory savings and a lower cache miss rate for normal users, thereby improving data access latency for them. As noted earlier, e-commerce websites experience a substantial proportion of malicious bot traffic, estimated at around 20% [18], [19]. INCS demonstrates a 78% accuracy in detecting bot traffic, which translates to improving the normal user data access experience by 15.6%, through the mitigation of bot-induced cache misses. The caching system's enhancements are expected to yield more substantial benefits for higher-level applications [11]. By limiting access requests from bots, INCS not only guarantees data access performance, but also reduces the amount of traffic to the backend.

In this work, we employed BMV2 on DPU for INCS implementation and to verify the functionality. More advanced switching solutions, such as OvS and the Data Plane Development Kit (DPDK), can potentially be leveraged too. As INCS is implemented using Planter [28], it can also be deployed on other targets such as programmable switches. However, this is not trivial, primarily due to limited processing and memory resources on switches. In a preliminary evaluation on Intel Tofino, a commodity programmable switch, INCS caching protection system achieved line rate (6.4Tbps) with sub-microsecond latency.

## VIII. Conclusion

In-network bot detection with machine learning represents a promising approach to safeguarding e-commerce ecosystems from malicious automated bot traffic. Through this study, we have explored a vulnerability of exploiting emerging programmable network devices to cache the hot items in the network. Furthermore, we have analyzed the characteristics of bot traffic and generated a dataset to evaluate this vulnerability as well as identifying it through in-network machine learning techniques. In conclusion, in-network bot detection with machine learning holds great promise in the fight against malicious bots. It offers an efficient approach to identify and mitigate bot-related threats. However, it is essential to remain vigilant, continuously improve detection techniques.

Future work will assess INCS in comparison with server-based bot detection models with respect to resource usage, data throughput, and response time. It will also explore the effect of hand-crafted optimizations on feature extraction and INCS performance overheads.

## IX. Acknowledgments

The research presented in this paper has benefited from the Experimental Infrastructure for Exploration of Exascale Computing (eX3), which is financially supported by the Research Council of Norway under contract 270053. This work was partly funded by VMWare. We acknowledge support from NVIDIA.

## References

[1] Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Sény Ndiaye Mbaye. Web scraping: state-of-the-art and areas of application. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 6040–6042. IEEE, 2019.

[2] Grażyna Suchacka, Alberto Cabri, Stefano Rovetta, and Francesco Masulli. Efficient on-the-fly web bot detection. *Knowledge-Based Systems*, 223:107074, 2021.

[3] Yang Luo, Guozhen She, Peng Cheng, and Yongqiang Xiong. Botgraph: Web bot detection based on sitemap. *arXiv preprint arXiv:1903.08074*, 2019.

[4] Kristof Stouthuysen. A 2020 perspective on "the building of online trust in e-business relationships". *Electronic commerce research and applications*, 40:100929, 2020.

[5] Venkatesh Shankar, Kirthi Kalyanam, Pankaj Setia, Alireza Golmohammadi, Seshadri Tirunillai, Tom Douglass, John Hennessey, JS Bull, and Rand Waddoups. How technology is changing retail. *Journal of Retailing*, 97(1):13–27, 2021.

[6] Imperva. How bots affect e-commerce. https://softprom.com/sites/default/files/materials/Imperva-Threat-Research-How-bots-affect-ecommerce-FINAL.pdf, 2019.

[7] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, 2017.

[8] Yuta Tokusashi, Hiroki Matsutani, and Noa Zilberman. Lake: The power of in-network computing. In *ReConFig*, pages 1–8. IEEE, 2018.

[9] Noa Zilberman. In-network computing. https://www.sigarch.org/in-network-computing-draft/, 2019.

[10] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[11] Asaf Cidon, Assaf Eisenman, Mohammad Alizadeh, and Sachin Katti. Cliffhanger: Scaling performance cliffs in web memory caches. In *NSDI'16*, pages 379–392. USENIX Association, 2016.

[12] Wenliang Su, Xiaoli Han, Hanlu Yu, Yiling Wu, and Marc N Potenza. Do men become addicted to internet gaming and women to social media? a meta-analysis examining gender-related differences in specific internet addiction. *Computers in Human Behavior*, 113:106480, 2020.

[13] Jing Qiu, Zhihong Tian, Chunlai Du, Qi Zuo, Shen Su, and Binxing Fang. A survey on access control in the age of internet of things. *IEEE Internet of Things Journal*, 7(6):4682–4696, 2020.

[14] Zafar Gilani, Reza Farahbakhsh, Gareth Tyson, and Jon Crowcroft. A large-scale behavioural analysis of bots and humans on twitter. *ACM Transactions on the Web*, 13(1):1–23, 2019.

[15] Grażyna Suchacka and Jacek Iwański. Identifying legitimate web users and bots with different traffic profiles—an information bottleneck approach. *Knowledge-Based Systems*, 197:105875, 2020.

[16] Xichen Zhang and Ali A Ghorbani. An overview of online fake news: Characterization, detection, and discussion. *Information Processing & Management*, 57(2):102025, 2020.

[17] Fuzhi Zhang, Yueqi Qu, Yishu Xu, and Shilei Wang. Graph embedding-based approach for detecting group shilling attacks in collaborative recommender systems. *Knowledge-Based Systems*, 199:105984, 2020.

[18] Eduardo Rocha. 2018 bad bot report: The year bad bots went mainstream. https://www.globaldots.com/resources/blog/2018-bad-bot-report-the-year-bad-bots-went-mainstream/, 2108.

[19] imperva. Bad bot report bad bots strike back. https://www.imperva.com/resources/reports/Imperva_BadBot_Report_V2.0.pdf, 2020.

[20] Stefano Rovetta, Grażyna Suchacka, and Francesco Masulli. Bot recognition in a web store: An approach based on unsupervised learning. *Journal of Network and Computer Applications*, 157:102577, 2020.

[21] Haitao Xu, Zhao Li, Chen Chu, Yuanmi Chen, Yifan Yang, Haifeng Lu, Haining Wang, and Angelos Stavrou. Detecting and characterizing web bot traffic in a large e-commerce marketplace. In *ESORICS'2018*, pages 143–163. Springer, 2018.

[22] Nicolas Poggi, David Carrera, Ricard Gavalda, Eduard Ayguadé, and Jordi Torres. A methodology for the evaluation of high response time on e-commerce users and sales. *Information Systems Frontiers*, 16:867–885, 2014.

[23] Takamasa Tanaka, Hidekazu Niibori, LI Shiyingxue, Shimpei Nomura, Hiroki Kawashima, and Kazuhiko Tsuda. Bot detection model using user agent and user behavior for web log analysis. *Procedia Computer Science*, 176:1621–1625, 2020.

[24] Muhammad Shafiq, Zhihong Tian, Yanbin Sun, Xiaojiang Du, and Mohsen Guizani. Selection of effective machine learning algorithm and bot-iot attacks traffic identification for internet of things in smart city. *Future Generation Computer Systems*, 107:433–442, 2020.

[25] Hongwen Kang, Kuansan Wang, David Soukal, Fritz Behr, and Zijian Zheng. Large-scale bot detection for search engines. In *WWW'10*, pages 501–510, 2010.

[26] Manos Tsagkias, Tracy Holloway King, Surya Kallumadi, Vanessa Murdock, and Maarten de Rijke. Challenges and research opportunities in ecommerce search and recommendations. In *ACM SIGIR Forum*, volume 54, pages 1–23. ACM New York, NY, USA, 2021.

[27] Changgang Zheng, Xinpeng Hong, Damu Ding, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. In-Network Machine Learning Using Programmable Network Devices: A Survey. *IEEE Communications Surveys & Tutorials*, pages 1–1, 2023.

[28] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Automating in-network machine learning. *arXiv preprint arXiv:2205.08824*, 2022.

[29] Changgang Zheng, Haoyue Tang, Mingyuan Zang, Xinpeng Hong, Aosong Feng, Leandros Tassiulas, and Noa Zilberman. DINC: Toward Distributed In-Network Computing. *Proceedings of the ACM on Networking*, 2023.

[30] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pForest: In-Network Inference with Random Forests. *arXiv preprint arXiv:1909.05680*, 2019.

[31] Mingyuan Zang, Changgang Zheng, Lars Dittmann, and Noa Zilberman. Towards Continuous Threat Defense: In-Network Traffic Analysis for IoT Gateways. *IEEE Internet of Things Journal*, 2023.

[32] Changgang Zheng, Benjamin Rienecker, and Noa Zilberman. QCMP: Load Balancing via In-network Reinforcement Learning. In *ACM SIGCOMM FIRA'23*, 2023.

[33] Xinpeng Hong, Changgang Zheng, Stefan Zohren, and Noa Zilberman. LOBIN: In-Network Machine Learning for Limit Order Books. In *HPSR'23*, pages 159–166. IEEE, 2023.

[34] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. IIsy: Practical in-network classification. *arXiv preprint arXiv:2205.08243*, 2022.

[35] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, et al. Scaling memcache at facebook. In *NSDI'13*, 2013.

[36] K Selçuk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal. Enabling dynamic content caching for database-driven web sites. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 532–543, 2001.

[37] Gerhard Hasslinger, Mahmoud Kunbaz, Frank Hasslinger, and Thomas Bauschert. Web caching evaluation from wikipedia request statistics. In *WiOpt'17*, pages 1–6. IEEE, 2017.

[38] Amazon. Database caching strategies using Redis. https://d1.awsstatic.com/whitepapers/Database/database-caching-strategies-using-redis.pdf, 2017.

[39] Alibaba Cloud. Redis hotspot key discovery and common solutions. https://www.alibabacloud.com/blog/redis-hotspot-key-discovery-and-common-solutions_594446, 2019.

[40] B Barla Cambazoglu and Ricardo Baeza-Yates. *Scalability challenges in web search engines*. Springer Nature, 2022.

[41] Pietro Michiardi, Damiano Carra, Sara Migliorini, et al. In-memory caching for multi-query optimization of data-intensive scalable computing workloads. In *EDBT/ICDT'19*, pages 1–8, 2019.

[42] Chen Luo, Vihan Lakshman, Anshumali Shrivastava, Tianyu Cao, Sreyashi Nag, Rahul Goutam, Hanqing Lu, Yiwei Song, and Bing Yin. Rose: Robust caches for amazon product search. In *Companion Proceedings of the Web Conference 2022*, pages 89–93, 2022.

[43] memcached. Replacing the cache replacement algorithm in memcached. https://memcached.org/blog/modern-lru/, 2018.

[44] Shangbin Feng, Herun Wan, Ningnan Wang, Jundong Li, and Minnan Luo. Twibot-20: A comprehensive twitter bot detection benchmark. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4485–4494, 2021.

[45] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, and Ali A Ghorbani. Towards effective feature selection in machine learning-based botnet detection approaches. In *2014 IEEE Conference on Communications and Network Security*, pages 247–255. IEEE, 2014.

[46] Wikimedia. Wikimedia rest api. https://wikimedia.org/api/rest_v1/#/, 2023.

[47] Amazon. Shopping queries dataset: A large-scale esci benchmark for improving product search. https://github.com/amazon-science/esci-data, 2022.

[48] Alibaba. e-commerce search benchmark. https://github.com/alibaba/eCommerceSearchBench, 2020.

[49] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *IEEE INFOCOM'99*, volume 1, pages 126–134. IEEE, 1999.

[50] Sunghwan Ihm and Vivek S Pai. Towards understanding modern web traffic. In *IMC'11*, pages 295–312, 2011.

[51] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *ACM SIGMETRICS/PERFORMANCE'12'*, pages 53–64, 2012.

[52] Qi Huang, Helga Gudmundsdottir, Ymir Vigfusson, Daniel A Freedman, Ken Birman, and Robbert van Renesse. Characterizing load imbalance in real-world networked caches. In *HotNets'14*, pages 1–7, 2014.

[53] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W Moore. Understanding pcie performance for end host networking. In *ACM SIGCOMM'18*, pages 327–341, 2018.

[54] Anuj Kalia, Michael Kaminsky, and David G Andersen. Design guidelines for high performance RDMA systems. In *USENIX ATC'16)*, pages 437–450, 2016.

[55] Guy Harrison and Michael Harrison. *MongoDB Performance Tuning*. Apress Berkeley, 2021.

[56] Behavioral model (bmv2). https://github.com/p4lang/behavioral-model, 2023.

[57] Ali AlSabeh, Joseph Khoury, Elie Kfoury, Jorge Crichigno, and Elias Bou-Harb. A survey on security applications of p4 programmable switches and a stride-based vulnerability assessment. *Computer Networks*, 207:108800, 2022.