INC: In-Network Classification of Botnet Propagation at Line Rate

Kurt Friday[1], Elie Kfoury[2], Elias Bou-Harb[1], and Jorge Crichigno[2]

[1] The Cyber Center for Security and Analytics
The University of Texas at San Antonio, USA
{kurt.friday,elias.bouharb}@utsa.edu
[2] Integrated Information Technology
The University of South Carolina, USA
{jcrichigno@cec, ekfoury@email}.sc.edu

Abstract. The ever-increasing botnet presence has enabled attackers to compromise millions of nodes and launch a plethora of Internet-scale coordinated attacks within a very short period of time. While the challenge of identifying and patching the vulnerabilities that these botnets exploit in a timely manner has proven elusive, a more promising solution is to mitigate such exploitation attempts at core traffic transmission mediums, such as within the forwarding devices of ISPs, backbones, and other high-rate network environments. To this end, we present an In-Network Classification (INC) technique to fingerprint the spread of botnets at wire-speed within busy networks. In particular, INC employs a unique bagging classification system residing entirely within programmable switch hardware in order to classify and subsequently mitigate bot infections amid Tbps traffic rates. Additionally, INC immediately pushes the data plane features of mitigated bots to the controller to infer botnet orchestration in real-time via behavioral clustering. INC was comprehensively evaluated against several datasets and achieved state-of-the-art results while reducing the detection times of comparable techniques by several orders of magnitude. Further, we demonstrate that INC can generalize well to previously unseen botnets.

Keywords: Bot detection · Botnet inference · Machine learning · Ensemble learning · Bootstrap aggregation · Bagging · Network traffic classification · Programmable switches · P4.

1 Introduction

Botnets have long been the key means of executing various Internet attacks, including Distributed Denial of Service (DDoS), spam, cryptojacking, identify theft, phishing, and more recently, ransomware. Despite numerous research efforts devoted to safeguarding modern-day networks against this coordinated maliciousness, botnet malware infections persist as one of the primary security concerns as outbreaks continue to escalate in both frequency and volume [1]. Indeed, it has been empirically demonstrated that a single botnet can infect upwards of

a million devices in a very short period of time [2]. Moreover, it is possible for enterprises to now be held liable for any attacks that their infected machines may conduct [6], despite the fact that such infections can occur without the network operator's knowledge [36].

A number of factors have amplified the spread of botnet malware; however, none of these factors are more fundamental to its propagation than the enhanced reachability of computing devices in general. Furthermore, the bots of a botnet typically have short lifetimes (e.g., a few days) due to patching procedures, malware detection mechanisms, etc. [31], so it is essential for a botnet to exploit this reachability to remain healthy. Thus, the most effective means of safeguarding the Internet at large against botnet infections is by eradicating the propagation of botnet malware at the Internet's core transmission mediums, namely, the data planes of ISPs, BackBones (BBs), Science DMZs, and other high-rate network environments. Additionally, it is paramount that detection is performed swiftly, given the speed at which botnet infections can propagate.

To this end, a number of botnet detection techniques employing network traffic analysis have been proposed; however, the state-of-the-art techniques still must be conducted at a time later than when the traffic was transmitted. Largely, these approaches are based on software that processes packet captures or flow data in an offline manner. Notably, software is incapable of processing the large traffic loads of the aforementioned high-rate network implementations without severe throughput degradation or even crashes [37]. Hence, software-centric approaches can not support broad-scale botnet detection or are tightly coupled with substantial detection latency.

Alternatively, a few approaches extract the features of traffic flows from the data plane's switches. As opposed to software, switch hardware is capable of handling high rates of network traffic. Then at a later predetermined point in time, the flow features are collected by a centralized machine (i.e., a controller) that fingerprints bot traffic. While this leveraging of the data plane is commendable, there are still research gaps that exist. In particular, these strategies rely on aggregating an abundance of flow data, which is both time consuming and taxing on a switch's limited storage. Moreover, the centralized detection may result in network congestion [32] and additional delays due to discrepancies between when the data collection for a malicious flow has completed in the data plane and when the detection is actually performed on the controller. Additionally, past implementations do not scale well to busy core networks that typically must process millions of unique IP addresses within very short periods of time and support asymmetric routing (i.e., bidirectional flow features cannot be utilized). Finally, while the aforementioned strategies fingerprint bots, they do not identify coordination between them. As a result, such strategies fail to provide key insights pertaining to specific services or previously-unknown vulnerabilities that should be hardened, future coordinated attacks that can expected and therefore provisioned for, etc.

To address these research gaps, we propose INC, an in-network classification scheme rooted in P4 [4] that not only extracts relevant features but also performs

bot detection entirely within the data plane via the programmable switch technology. In particular, INC instruments such switches with an Adapted Switch Classification (ASC) technique, which circumvents the aforementioned pitfalls associated with centralized detection strategies. ASC was constructed in a compact manner for high-throughput switches (supporting up to 25 Tbps traffic processing [13]) and to run alongside a variety of other essential switch applications. To the best of our knowledge, INC is the first full-fledged, switch-based classification methodology (i.e., the extraction and subsequent storing of features, followed their classification) that is conservative enough with the switch's limited resources to allow the concurrent execution of switch's required forwarding and telemetry applications. Further, INC requires a small number of unidirectional packets from a source IP address to detect if it is a bot, which enables both asymmetric routing implementations and detection speeds that are many orders of magnitude faster than the state-of-the-art. Additionally, INC is equipped with a novel data structure for the storing of granular per-IP features necessary for accurately fingerprinting individual bots amid the millions of IPs that busy core network environments process in short time windows. Lastly, INC transmits the features of bots upon detection to the centralized controller in order to attribute the presence of botnet campaigns in real time.

INC was comprehensively evaluated amid a variety of noteworthy datasets. We demonstrate that INC not only achieves an F1-score of 0.99 for bot classification on the test split of the training data but also performs similarly on botnet datasets it has not observed. We also conducted a comparison with state-of-the-art approaches that rely on features extracted from the data plane, certifying that INC performs similarly or better than such resource-intensive, centralized Machine Learning (ML) classifiers. Finally, we show that INC can also detect the presence of botnets with an accuracy of 99%.

The core contributions of this work are highlighted as follows:

- We offer INC as a first-of-a-kind means of mitigating botnet propagation and inferring orchestrated botnets at wire-speed via the newfound programmable switch technology. INC was specifically tailored towards busy network implementations to address the botnet epidemic at scale and to fingerprint such maliciousness within a fraction of the time of past approaches. To facilitate future cybersecurity advancements in this domain, we make all source code publicly available, as well as a variety of code for conducting associated data analysis tasks [19].
- We propose a novel conversion of the Decision Tree (DT) data structure (i.e., ASC) to allow it to be placed entirely within the switch with sufficient resources to spare for several other data plane applications. To the best of our knowledge, this has never before been achieved. Additionally, practitioners can leverage ASC to perform a variety of other ML applications on the switch.
- We perform a comprehensive evaluation of INC on several notable datasets and show that INC is, at minimum, as effective as the state-of-the-art ML approaches for bot detection and similarly effective on unseen data.

The remainder of the paper is organized as follows. In the following section, we begin by introducing the threat model for this work. In Section 2, we present INC and detail its various intricacies. Subsequently, Section 3 encompasses a comprehensive evaluation of INC to verify its effectiveness and compare its performance to the state-of-the-art. In Section 4, we review the related literature. Finally, we conclude this work and summarize its findings in Section 5.

1.1 Threat Model

We begin by first elaborating upon the threat model for which INC was designed. Ultimately, our threat model is based on modern and prominent botnet malware. In general, such malware may explicitly target Internet of Things (IoT) devices, typical environments such as workstations, or both. In turn, we do not make any assumptions about these botnets' modus operandi, other than it they will have to emit network traffic to achieve their aim. This traffic may include varying degrees of propagation attempts and exchange of information with a Command and Control (C&C) server. We address these variations in the traffic in different strains of contemporary botnet malware by only considering a particular sequence of packets to be malicious if it contains some attempt at propagation or C&C contact. This technique accounts for the fact that infected nodes will often still be performing their typical duties and therefore be transmitting benign traffic as well. Such a sequence of packets is referred to as p_{thresh} herein.

In terms of deployment, the switch in which INC resides must be on the path of the aforementioned malicious traffic in order to detect it. For example, an ISP could protect its business and residential consumers by placing INC on the paths utilized by their traffic, as these entities are typically associated with the vulnerable IoT devices and workstations that botnets exploit. Whether such paths with the highest coverage are within the core layer or elsewhere can very depending upon the given network fabric. That said, the small p_{thresh} number of packets that INC requires to accurately fingerprint a bot circumvents the need to consider collaborative detection with choosing a placement strategy. Furthermore, with INC's reliance upon a small p_{thresh} and no recirculations, any forwarding scenarios for a given topology where one INC-empowered switch might observe some of the same traffic as another will have no impact on classification performance. Finally, INC does not require controller aid to perform detection or bidirectional flow analysis. Indeed, the aforementioned advantages coupled with the cost benefits of leveraging programmable switches [37] enable INC to support any number of placement options and strategies.

2 Proposed Approach

In this section, we present INC's methodology for detecting isolated bots and performing botnet campaign inference. In particular, an overview of the proposed approach is discussed to offer intuition about its modus operandi. Next,

the four primary components of INC are highlighted, namely, the Register Recycling Manager (RRM), feature aggregation, ASC, and botnet inference. Fig. 1 visualizes these components (denoted with blue) and INC's overall flow of execution. We kindly refer interested readers to our publicly available source and data analysis code [19] for additional details pertaining to the implementation of INC.

2.1 Overview

When a machine is infected by a given botnet's malware, aside from the wide variety of malicious acts that the bot might be commanded to perform, there are generally three types of traffic that it may transmit: (1) probing, (2) C&C-related communications, and (3) traffic associated with the unsuspecting user or any benign processes running on the machine. Thus, the sooner that traffic types (1) and (2) originating from the source IP IP_{source} of a bot can be fingerprinted, the faster any further propagation of the botnet can be mitigated and any future malicious acts that it may perform can be prevented. To this extent, INC aims to dramatically reduce number of packets from a particular IP_{source} that are required to detect traffic types (1) and (2). As portrayed in Fig. 1, this aim is achieved by ASC's classification on features aggregated by INC's Feature Aggregation component over a reduced window of p_{thresh} consecutive packets from such an IP_{source}.

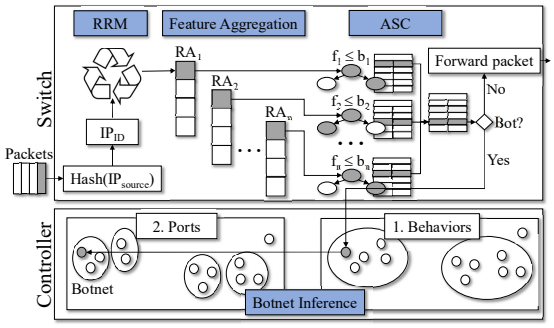


Fig. 1: Overview of INC. The primary components are highlighted in blue, which correspond to subsections in Section 2.

The challenge with storing the aforementioned aggregated features associated with every IP_{source} in a high-rate network environment that observes many IP addresses is that the switch does not house nearly enough SRAM storage to support such an endeavor. For instance, even if slots of SRAM could be allocated for the feature aggregation of, say, 200K different IP_{source} values, that is still far less than would be needed to account for the millions of IP_{source} values that could arise in backbone environments. INC mitigates this issue with the RRM, which promptly frees-up such limited slots so the switch can begin aggregating the features of newly-arriving IP_{source} values immediately; otherwise, the latency before a newly-arriving bot IP_{source} receives a slot may allow further infections and other maliciousness to take place prior to detection.

Once the p_{thresh} window for an IP_{source} has concluded, ASC endeavors to classify whether traffic types (1) and (2) are present in the window of packets via the via the aggregated features, in order to determine if the IP_{source} is a `bot` or `benign` entity. However, since novel botnets are increasingly surfacing

and their techniques for performing malicious behavior are constantly evolving, an ML model trained on a few botnets can quickly become obsolete. As such, a primary aspiration of ASC is to generalize well to botnet data that was not observed during the training process. In turn, ASC employs a bagging technique based off shallow DT learners that are learned in parallel via bootstrapping and their predictions act as votes towards classifying whether an IP_{source} is a `bot` or `benign`. Since the DT learners are shallow, they offer a dramatic reduction in variance, which promotes ASC's ability to generalize to botnets that it was not trained on. However, the combination of the learners being shallow and that DTs are trained in a greedy fashion can lead to bias. To address this issue, we propose a feature selection technique that mitigates such bias by identifying aggregated features that are ideal for binary classification and generalization to unseen data. In the event ASC classifies an IP_{source} as a `bot`, the IP_{source} is considered mitigated as its traffic can be easily blocked or monitored at the discretion of the network operator. Additionally, the bot's aggregated features are immediately pushed to the controller to attribute the IP_{source} to a botnet, as portrayed in Fig. 1. At this juncture, the controller performs a behavioral clustering procedure on such features with those from other attributed bots to fingerprint occurrences of coordination. Indeed, such behavioral analysis is a strong indicator of orchestrated botnets [22].

2.2 RRM

The RRM is positioned within INC's flow of execution as shown in Fig. 1 and has the primary task of mitigating any bot detection delays for a newly-arriving bot's IP_{source} due to all SRAM slots being occupied. The RRM addresses this issue by freeing-up a slot on an as-needed basis so that INC can begin aggregating such a bot's features over its p_{thresh} packet window. In a similar vein, the freeing-up of slots on an as-needed basis also promotes utilizing a smaller pool of slots to consume less SRAM registers within the switch. As portrayed in Fig. 1, an IP_{source} is first hashed before reaching the RRM. The hashing is performed by a 32-bit Cyclic Redundancy Check `CRC32` algorithm, as CRC is computationally inexpensive and readily available in many high-speed networking devices, particularly in Intel Tofino switches [12]. This `CRC32` hashing procedure returns an array index IP_{idx} that enables INC to map each IP_{source} to its corresponding features that are stored in SRAM Register Arrays (RAs). The set of *all* registers located at IP_{idx} within such RAs act as the slot for a given IP_{source}.

Collisions. Utilizing a smaller pool of slots can result in additional IP_{idx} collisions. However, the `CRC32` hashing algorithm uniformly distributes each IP_{idx} throughout the RAs (as opposed to using a portion of the IP_{source} as an IP_{idx}) and thereby reduces such collisions. A collision can result in a benign source being wrongly fingerprinted as malicious (a false positive) or even allowing a malicious source to appear benign (a false negative). The only guaranteed way of negating collisions entirely is for INC to store the IP_{source} currently using every IP_{idx} to compare against the IP_{source} of any incoming packets with the same IP_{idx}. Thus, we employ such a measure via storing each IP_{source} at index

IP_{idx} within RA_{IP}, as shown in the RRM design portrayed in Fig. 2. This figure follows the convention of utilizing a subscript p for data associated with the previous IP_{source} held at IP_{idx} and a subscript c for that of the newly-arrived IP_{source} that is currently being processed. Note that all RAs, both within the RRM and the subsequent Feature Aggregation component shown in Fig. 1, have a predetermined size RA_{size}, which is later evaluated in Section 3.4.

Priority groups. To free-up a slot for an incoming IP_{source}, Priority Groups (PG) are integrated into the RRM to determine when to recycle the RAs' registers located at IP_{idx} for this IP_{source}. As shown in Fig. 2, the PG has a bitmap array at every IP_{idx}, which denotes whether a previous IP_{source} that hashed to such IP_{idx} has been previously classified as benign by assigning the IP_{source} a priority value of 1 (as opposed to a higher priority of 0). The RRM also maintains a *timestamp* RA to control how long an IP_{source} should be allowed to occupy a slot in the event it has a priority value of 0. Additionally, note in Fig. 2 that `CRC32` is applied as opposed to `CRC16` because five additional bits are needed to identify the PG index at the given IP_{idx} RA location.

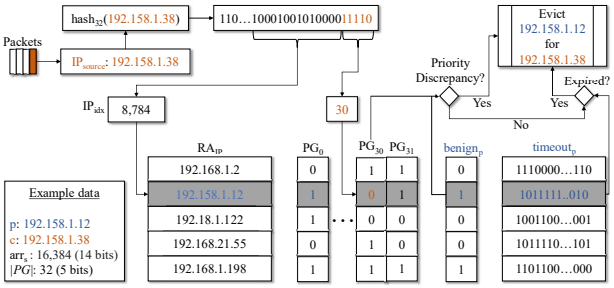


Fig. 2: RRM design.

Fig. 2 visualizes an example operation of the RRM. In this example, IP_{source_p} 192.158.1.12 has been blocking IP_{idx} 8,784 while its features have been aggregating until its p_{thresh} packet window has been reached to trigger classification. However, another packet with an IP_{source_c} 1192.158.1.38 sharing the same IP_{idx} of 8,784 has just arrived. Since $IP_{source_p} \neq IP_{source_c}$, IP_{source_c} will replace IP_{source_p} and all other RA locations at IP_{idx} will be recycled for IP_{source_c}, given IP_{source_c} has a higher priority of 0. Note that if there was no such priority discrepancy, IP_{source_c} would still be evicted for IP_{source_p} if $timestamp_p$ was expired.

2.3 Feature Aggregation

As depicted in Fig. 1, once the RRM has established the IP_{source} to be considered, the switch then performs the aggregation of its features and stores them at IP_{idx} within the feature RAs. Prior to selecting a final feature set that INC will be tasked with aggregating per-IP_{source}, we first formulate a pool of potential features. Note that since programmable switches cannot inspect packet payloads without performance penalties [14], INC only aggregates features that can be derived from network and transport layer headers, as well as metadata such as timestamps, using simple operations that switches can easily support. Thus, we applied either counts or summations to a particular header field h over

p_{thresh} to arrive at such a pool. More formally, we employ $\sum_{n=1}^{p_{thresh}} h_n$ for summations and $\sum_{n=1}^{p_{thresh}} f_h(h_n)$ for counts, where header function $f_h(h_n)$ returns a value of 1 or 0 depending on whether a condition on h_n is met or not. In general, header fields that do not generalize well (e.g., IP addresses) were not considered as features. An exception was a feature we devised called `alternating_dst`, where $f_h()$ receives the h_n of destination IP d_ip_n within packet n and returns 1 when $d\_ip_n \neq d\_ip_{n-1}$. Indeed, `alternating_dst` *may* indicate that an IP_{source} is probing for vulnerable nodes to infect or endeavoring to contact its C&C server.

Subsequently, we aim to select the features from the aforementioned generated pool that will generalize well to unseen data when applied to ASC's shallow learners. To this end, we identify each feature g_j whose discrete Probability Density Function (PDF) $f_{botnet_i}(X_{g_j})$ for a botnet dataset i of data samples X has a maximum amount of overlap with other botnet datasets and similarly has a PDF $f_{benign_i}(X_{g_j})$ for a benign dataset i with a maximum amount of overlap with other benign datasets. At the same time, such overlap should be minimized between all f_{botnet_i} and f_{benign_i} to promote binary classification performance. Recall that a g_j value for a given data sample x is actually an aggregation of this g_j value over p_{thresh} consecutive packets.

We define this measure of overlap $\eta(f_1(), f_2())$ between two discrete PDFs $f_1()$ and $f_2()$ as $\eta(f_1(), f_2()) = \sum_{k=1}^{|bins|} min(f_1(x_k), f_2(x_k))$, where η falls within the range $[0, 1]$. The X inputs are divided among equally-distributed bins ranging from the first to last input x of $f_1() \bigcup f_2()$ and set $|bins| = min(num\_inputs, 1024)$, where num_ints is the number of such x inputs in $f_1() \bigcup f_2()$. Note this overlap measure does not make any assumptions about underlying feature distributions, which can limit the use of this measure in practice [21].

To arrive at the final η for each feature g_j, arithmetic means are taken for each of the $\binom{n}{2}$ $(f_1(), f_2())$ pairs in the botnet and benign datasets to arrive at η_{botnet} and η_{benign}, respectively. Next, a minimal $\eta(f_{\bigcup botnet}(), f_{\bigcup benign}())$ is sought after, where $f_{\bigcup botnet}()$ and $f_{\bigcup benign}()$ are the unions of the g_j distributions within the botnet and benign datasets, respectively. We set $\eta_{final} = {}^{\eta_{botnet}}/_4 + {}^{\eta_{benign}}/_4 + {}^{1-\eta(f_{\bigcup botnet}(), f_{\bigcup benign}())}/_2$, and select the largest m η_{final} values to obtain m features to apply towards ASC's training. Note that η_{final} was formalized in this manner, since η_{botnet} and η_{benign} share equal importance regardless of the number of datasets encompassed by each, and the maximization and minimization operations should also be treated equally.

2.4 ASC

As portrayed in Fig. 1, features processed by INC's Feature Aggregation component are subsequently leveraged by ASC to classify a given IP_{source} stored at IP_{idx}. ASC is based on a methodology for transforming Decision Tree (DT) classifiers to programmable switch data structures that perform classification in two stages. This methodology is in stark contrast to performing DT traversals sequentially, which exhausts numerous stages within the switch and therefore is

not practical or even feasible in many scenarios. To offer some intuition, a simplified overview of this transformation using only two features is given in Fig. 3. After the controller performs training of a decision tree, it uses the values that the features are compared against (x_i and y_i in Fig. 3) within the DT's nodes as decision boundaries for each feature. The Range Interpretation of Fig. 3 gives a visualization of how ultimately these decision boundaries divide shaded ranges that translate to classifications.

To arrive at such classifications on the switch, the first stage of ASC is used to identify within which decision boundary-based ranges the features reside. This is achieved by applying a Match-Action Table (MAT) for each feature in order to map it to an integer that uniquely identifies a given feature's range, as shown in Fig. 3. Additionally, note that there are no dependencies between features utilizing ACS's methodology, and therefore these feature tables are executed in parallel within the first stage. As demonstrated in Fig. 3, a classification MAT in the second stage is applied that performs an `exact` match on each returned integer from the feature tables and outputs a corresponding classification.

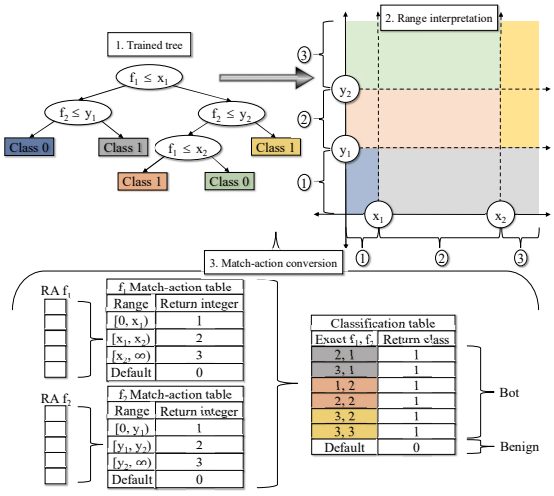


Fig. 3: ASC's parallel execution strategy.

Since ASC performs binary bot detection, observe that it also reduces the number of classification MAT entries by only matching upon the integers that translate to a single class and letting any other input that does not match `Default` to the other class. When extending ASC to multiple DTs within a bagging ensemble, the ranges of each feature are combined over all DTs, ensuring that there are no overlaps between ranges so each combination of the feature MATs' return integers correspond to only one entry in the subsequent classification MAT. For instance, if an overlap of two ranges for a particular feature is identified, the overlap is split into three adjacent ranges. To evaluate ranges in P4 MATs, `ternary`, `lpm`, or `range` match keys can be leveraged. Indeed, the embedding of ASC's methodology within MATs allows the classifier to be updated on the fly, such as when new botnet intelligence arrives, because the controller can populate each of the MATs with the match keys and return values during runtime.

In order to train ASC effectively, we first extract consecutive packets for different p_{thresh} values from each benign and botnet IP_{source} within the datasets covered later in Table 1. As infected machines often transmit legitimate traffic as well, we only label a p_{thresh} sequence of packets originating from a bot

IP_{source} as malicious if it contains at least one packet tied to the given botnet malware (e.g., a packet that is probing for vulnerabilities, executing malicious commands, attempting to contact a C&C server, etc.). Once such labeling was performed on the datasets, ASC was trained on the controller via Gini impurity and the features identified by leveraging the aforementioned feature selection strategy proposed in Section 2.3. Additionally, `Grid Search` was executed on the number of shallow learners in the bagging ensemble and their depth, with the aim at minimizing such depth for generalization purposes. Ultimately, utilizing weighted decision stumps for the features `syn_flag_count`, `ack_flag_count`, `tcp_flag_sum`, `alternating_dst`, `eth_size_sum`, and `ip_id_sum` gave the ideal trade-off between classification performance and a minimal depth of the shallow learners. Note that weighting is accounted for by the controller when populating the classification MAT, and therefore ASC's P4 implementation requires no modifications in order to integrate such weighting.

2.5 Botnet Inference

The controller performs a two-phase clustering procedure on the aggregated features of a bot that are pushed by the switch immediately after it is detected, which is shown in Fig. 1. Such procedure is executed at each interval I, with the length of I being dependent upon the controller's available resources and time constraints of the given implementation. Recall that at this juncture, any propagation attempts by the infected IP_{source} have been halted by the switch. HDBSCAN [18] is leveraged for both clustering phases, as it shares many of the core advantageous of hierarchical clustering and its predecessor, DBSCAN [28], yet has undergone several optimizations. Such optimizations include an ability to work with arbitrarily shaped clusters of varying sizes and densities, and enhanced speed. Phase-one entails clustering the bots' behaviors based on the aggregated features used by ASC for classification. Alternatively, the phase-two clusters each of the phase-one clusters by destination ports. To provide the ports for the phase-two clustering, the switch stores the top three most targeted ports by every IP_{source}, which necessitates three more RAs, respectively. Note that these ports are note applied to bot classification by ASC, since the IP_{source} has been classfied as bot, it is expected that the top ports will correspond to the services that the bot is leveraging for malicious acts. Past research has empirically demonstrated that most botnets will target between one and five ports [9,34]. Therefore, three ports is reasonable to capture the breadth of the port targeting behavior of an IP_{source} yet conserves the switch's SRAM. The end result of this two-phase clustering procedure are robust clusters of botnet campaigns.

3 Evaluation

In this section, we evaluate INC extensively to asses its efficiency and effectiveness. Results are reported by way of confusion matrices and accuracy, as well as F1-scores to measure classification results with imbalanced data.

Testbed. The environment setup consisted of four Intel Xeon Silver 4114 machines. Each machine functions off 32 CPUs at 2.20GHz, underneath Debian GNU Linux 9. Additionally, an Edgecore Wedge 100BF-32X [20] switch programmed with P4 was employed to forward traffic between the four machines. The switch was designed for high-performance data centers with programmable Tofino switch silicon and encompasses 32x100 Gbps ports which equate to a throughput processing capacity reaching 3.2 Tbps. The datasets utilized for the following experiments are detailed in Table 1, where a synopsis of each is given.

Source	Description	Size
Benign P2P [23]	A large group of benign packet captures of the P2P applications Skype, eMule, FrostWire, μTorrent, and Vuze	62 GB
General benign data [30]	An aggregation of 18 benign packet captures including a variety of traffic, such as HTTPS interactions, DNS requests, and P2P.	9 GB
CAIDA BB traffic [5]	A large set of packet captures containing anonymized Internet BB traffic from CAIDA	50 GB
Mirai botnet [11]	Packet captures from a medium-sized IoT network infrastructure consisting of 83 IoT devices	813 MB
Bashlite botnet [11]	Packet captures extracted from a medium-sized IoT network infrastructure consisting of 83 IoT devices	531 MB
Trickbot botnet [30]	Three packet captures corresponding to three Trickbot-infected machines targeting both non-IoT and IoT devices	452 MB
Dridex botnet [19]	Five packet captures from five variations of the Dridex botnet malware, respectively, that were executed in a Triage sandbox [24] consisting of non-IoT devices	261 MB
Emotet botnet [19]	Four packet captures from four variations of the Emotet botnet malware, respectively, that were executed in a Triage sandbox [24] consisting of non-IoT devices	319 MB
Neris botnet [10]	Packet captures extracted from 10 non-IoT devices encompassing benign traffic and Neris botnet samples utilizing IRC, ClickFraud, and SPAM	97 MB Neris, 52 GB benign
CAIDA SIP-scan [8]	Samples of each UDP probing packet targeting port 5060 captured by the CAIDA /8 network telescope	425 MB

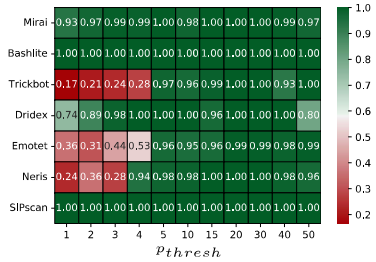
Table 1: Synopsis of the incorporated datasets.

The underlying motivation for selecting these botnets was to establish a representative sample of both IoT and non-IoT botnets. For instance, Mirai and Bashlite were chosen as IoT botnets because the majority of IoT botnets were found to be derived from them [7]. The controller performed ASC's training and subsequent deployment to the Tofino hardware switch. Next, the packet captures listed in Table 1 were replayed by three Linux machines through the Tofino switch using `tcpreplay`, which forwarded the packets to the fourth Linux machine that doubled as both the destination entity and controller. The primary function of the the fourth Linux machine was to aggregate the results of the experiments. Ultimately, as we are evaluating the unidirectional classification capabilities of INC in a high-rate forwarding environment, the environment was designed to transmit the packet captures though the switch simultaneously, while ensuring the interarrival times of the packets within the captures were respected.

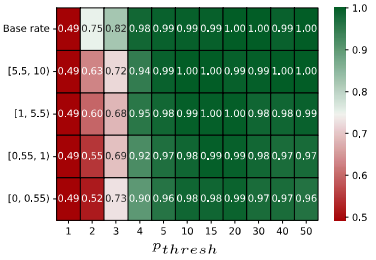
3.1 Bot Detection

pkt_{thresh} **assessment.** A small pkt_{thresh} allows INC to dramatically reduce detection latency, improve RRM register recycling, and circumvent the overhead as-

sociated with network-wide traffic statistics aggregation. In turn, we first endeavored to identify the optimal pkt_{thresh} value. To this end, it should be noted that infected hosts may also have a number of benign processes transmitting traffic. Moreover, multiple hosts behind Network Address Translation (NAT), say, one infected host and several benign, may appear as one IP to a switch on an external network. Thus, to evaluate INC's ability to handle such circumstances, we also interleaved with the collection legitimate benign traffic [30] mentioned in Table 1 at different interarrival time ranges, namely, [0.0, 0.55), [0.55, 1.0), [1.0, 5.5), and [5.5, 10.0) milliseconds. Subsequently, once ASC was trained on 70% of data, an evaluation of pkt_{thresh} over these interarrival time ranges was performed. The F1-score achieved by INC, along with the `Base rate` with no additional benign traffic interleaved (several botnet datasets already have intermittent benign traffic), is offered in Fig. 4.a. It can be observed that while there is subtle performance degradation with increased rates, the F1-scores still remain relatively consistent, with $p_{thresh} = 20$ giving the best performance.



(a) Interleaved benign traffic.



(b) Unseen botnet generalization.

Fig. 4: Evaluating INC amid varying p_{thresh}.

Generalization to unseen botnets. Given that new botnets are consistently surfacing, INC should also be able to effectively generalize to such unseen botnet traffic. To asses INC's ability to do so, we iteratively removed one botnet dataset from the training data and then tested the trained ASC model on the botnet dataset that was held out. Since there is only one label in the test set in this scenario, accuracy was utilized to depict INC's performance in Fig. 4.b. As shown, while the less aggressive nature of Trickbot and Neris gave INC trouble for the first four pkt_{thresh}, whereas INC generalized near perfectly (rounding is performed for visualization purposes) by $pkt_{thresh} = 20$.

3.2 Botnet Inference

We substantiated INC's ability to attribute bots to botnets by evaluating the clustering performance of the first 20K bots for each botnet that were fingerprinted by ASC in Section 3.1. Additionally, we utilized $pkt_{thresh} = 20$ as it gave the best performance and the `Base rate` to ensure a more controlled experiment. The results of this evaluation are given in the Table 2. As shown, the majority of the bots were attributed to their respective botnet. The bulk of the misclassifications were associated with Mirai, which were largely due to Mirai's high-variance behavior falling in between that of Bashlite and Trickbot. For example, the mean interarrival time for Mirai was approximately 3.165 seconds,

Mirai	Bashlite	Trickbot	Dridex	Emotet	Neris	SIPscan	Outliers
19,524	417	0	0	0	0	0	159
0	20,000	0	0	0	0	0	0
0	0	19,669	49	177	7	0	78
0	0	52	19,911	6	0	0	31
0	0	23	40	19,908	16	0	13
0	0	0	8	16	19,968	0	8
0	0	0	0	0	0	20,000	0

Table 2: Botnet Inference Results

versus roughly 0.000041 and 40.01 seconds for Bashlite and Trickbot, respectively. To add to the confusion, both Mirai and Bashlite heavily target Telnet port 23, which is why a number of Mirai samples were clustered as Bashlite. Recall that INC's phase-two clustering is strictly performed on destination ports, which is why Mirai's other misclassifications were considered outliers since Bashlite is the only other botnet that shares a common destination port. A similar occurrence can be observed between the botnets that rely on non-IoT devices due to their interaction with destination port 443. That being said, INC still achieved an overall accuracy of 0.9927 for this evaluation.

3.3 Comparison with Existing Solutions

While there are no other bot detection techniques that solely use the data plane, we identified three notable works to compare INC against that apply features extracted from the data plane to controller-based classifiers. The first of such research efforts was the recently-proposed FlowLens [3]. FlowLens extracts quantized sequences of both interarrival times and packet lengths as features for control plane-based ML algorithms. Within their use cases, the authors demonstrated the ability of such features to enable botnet chatter detection when using a Random Forest (RF) running on the controller. The second approach we compared INC against is coined EDIMA (Early Detection of IoT Malware network Activity) [16]. EDIMA relies on header-based features that can be applied to traditional data plane applications, namely, the number of unique destination IP addresses and the minimum, maximum, and mean of the number of packets sent to each destination by a particular source IP. EDIMA also applied an RF, as well as a K-NN and naive Bayes classifier. The final work we compared INC to is a technique presented by Letteri et al. [17] that inputs 22 data plane-based traffic features into a Deep Neural Network classifier (DNN). Note that all of these works were geared towards identifying the presence of bots on a network and not coordination between them (i.e., botnets); therefore, we only compare these works to our ASC implementation.

The best implementation of each of the aforementioned works was used for this evaluation, as specified in each paper. Two comparisons were then performed for all models: (1) a performance evaluation on the test split of all datasets and (2) an assessment of how well each model can detect botnets it has not observed during training. Specifically, *all* datasets listed in Table 1 were trained upon for comparison (1), and the same setup that was applied to INC's generalization evaluation in experiment 3.1 was utilized for comparison (2). Regarding both FlowLens and EDIMA, an RF with 1024 trees gave

the best results and was therefore applied. For the DNN presented by Letteri et al. [17], seven hidden layers containing 44, 88, 176, 88, 44, 22, and 11 neurons, respectively, were utilized. The Adam optimizer [15] was also employed for training this DNN, along with a learning rate of 0.001, batch sizes of 100, and ten epochs. The results for comparison (1) are listed in Table 3. As can be observed, ASC outperformed the other more complex ML algorithms running on the controller for the observed data, which we largely attribute to the features that INC leverages. Timing comparisons were not made as the other works all require the control plane for classification, and no details were given pertaining to when the controller executes the ML algorithms (e.g., running batch classifications upon expiring time windows, etc.).

Work	Observed F1	Unseen F1
FlowLens [3]	0.9818	0.0000
EDIMA [16]	0.8870	0.0000
Letteri et al. [17]	0.6677	1.000
INC (our approach)	**0.9907**	**0.9961**

Table 3: State-of-the-art bot detection comparison results.

Comparison (2) revealed a much larger disparity in each model's performance when fingerprinting bots whose data was not previously trained upon, as displayed by the mean F1-scores in Table 3. Interestingly, both FlowLens and EDIMA performed poorly. This is notable when considering that these state-of-the-art techniques leverage RFs, yet INC's decision tree-based approach achieved good results; thus, it is clear that this degradation of accuracy is not due to FlowLens and EDIMA's model selection, but rather their data preprocessing. This preprocessing entails aspects such as the length of the flow duration required (e.g., an hour in the case of FlowLens), features utilized, and so forth. Alternatively, it appears that the DNN implementation proposed by Letteri et al. performed markedly well at first glance; however, it is apparent from Table 3 that such performance is due to this model's tendency to classify all samples as `malicious`.

3.4 Deployment Microbenchmarks

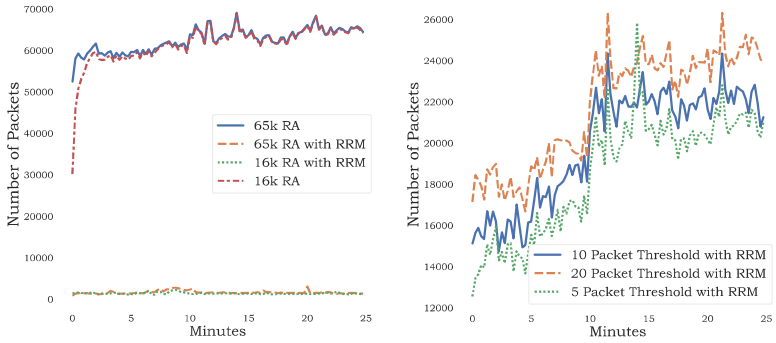
Hardware resource usage. The resource utilization of INC when compiled on the Tofino switch is listed in Table 4. As can be observed, INC consumes very little resources. For instance, the TCAM utilization which is leveraged for the ASC's parallel feature MATs only consumed 3.3% of that available. Moreover, the entirety of INC's operations occupied only four stages, which allows a number of other P4 applications to run alongside it. Finally, INC maintained a low SRAM consumption of 10.00%, which we attribute to the RRM enabling the use of small RAs.

Hash bits	Gateway	SRAM	ALU	TCAM	Stages
10.48%	12.50%	10.00%	0.00%	3.33%	4.0

Table 4: Tofino hardware resource utilization.

RRM Analysis. To effectively evaluate the RRM implementation, we assessed over 50 GB of real BB traffic provided by CAIDA [5] for 25 minutes. The results

from this analysis are shown in Fig. 5.a and Fig. 5.b. Note that smoothing was applied to enhance the visualization of trends. Fig. 5.a displays the number of collisions exhibited when applying different size RAs for storing each IP_{idx}, namely, an RA containing 65,536 registers and another encompassing 16,384.



(a) RRM collisions. (b) Missed packets.

Fig. 5: Evaluating RRM amid BB traffic.

Additionally, both sizes were evaluated both with and without the RRM. As shown, the reduction in the number of collisions that the RRM provides for both sizes is dramatic. Additionally, both the 16,384 and 65,536 RAs without RRMs gave a similar amount of collisions. Thus, we deem RA sizes of 16,384 effective for high-rate network implementations.

Alternatively, Fig. 5.b displays the number of packets that were not analyzed due to their given RA index IP_{idx} already being occupied by another IP_{source}. With INC striving to minimize p_{thresh} to reduce this occurrence, we evaluate three different p_{thresh} values on the BB trace. As Fig. 5.b portrays, decreasing the packet count can reduce the number of such packets not analyzed by several thousand. Additionally, note that the *timeout* was set to a very high 30 minutes and is the cause of the increase over time of packets that were not analyzed. This is because if the amount of packets that an IP_{source} transmits falls under p_{thresh}, it will not be discarded and will continue to occupy the index IP_{idx} of the RAs until such *timeout* is reached. Nevertheless, there are about 39 million distinct packets recorded per every 0.1 seconds in this trace, and therefore the RRM roughly obtained a missed packet rate falling within the range of [0.036%, 0.062%].

Examining real-world applicability. We additionally evaluated INC in real-world conditions by examining the hardware performance amid high-rate traffic. Specifically, we performed two experiments with one server to transmit the botnet datasets listed in Table 1 along with two servers to send the aforementioned CAIDA BB trace through the switch. The first of the experiments entailed transmitting

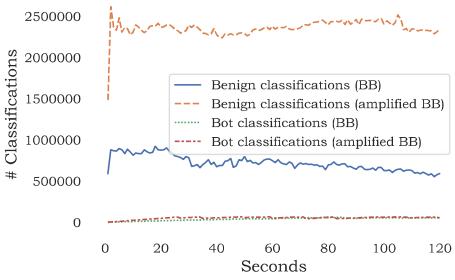


Fig. 6: Tofino hardware BB assessment.

both the botnet and CAIDA traffic at their captured rates (arriving at an average of over 2 Gbps), whereas the second sent three separate instances of the CAIDA trace (totaling upwards of 6 Gbps on average) while keeping the botnet traffic at its captured rates. As can be observed in Fig. 6, the number of benign classifications of the BB traffic was consistent with the increased rates. Moreover, Fig. 6 demonstrates that the number of bot classifications remains relatively fixed regardless of the increased BB. Ultimately, INC gave an average F1-score of over 0.99 in both experiments, which demonstrates its ability to generalize well to a variety of benign BB traffic. Note that the time for each packet to be processed by INC was also measured, which was fixed at approximately 350 nanoseconds amid both experiments.

4 Related Work

The commendable efforts by Barradas et al. [3], Kumar and Lim [16], and Letteri et al. [17] that INC was compared against in Section 3.3 are the approaches in the literature that were able to leverage data plane-based features for controller-based bot detection with ML algorithms. However, a growing area of research has been the synergizing ML techniques with the programmable switches themselves. For example, the processing power of such switches has been leveraged to speed up the computational overhead of ML tasks [25–27, 35]. Alternatively, Siracusano et al. [29] took a noteworthy first-step towards implementing neural networks in P4 by way of utilizing only the bitwise logic programmable switches can support, such as quantized binary weights. Subsequently, Xiong and Zilberman [33] introduced theoretical strategies for integrating more traditional ML algorithms into the switch's pipeline. By the authors' admission, it is unclear as to whether actual switch hardware can support their implementations, as they are resource intensive.

5 Conclusion

Botnets and their prompt propagation have enabled adversaries to cause considerable damage to contemporary networks, often before administrators are even aware that an infection has taken place. However, the emergence of programmable data planes finally offers a viable means of countering this ongoing maliciousness. To this end, we presented INC, an ML-based approach rooted in programmable switches to detect and mitigate botnet propagation at scale, in real-time. By way of employing several notable datasets, we demonstrated that INC can not only detect a variety of botnet malware with an F1-score upwards of 0.99 on average, but it can also both fingerprint unseen bots and attribute coordination between them with negligible performance degradation. Finally, we showed that INC resource-conservative approach can outperform state-of-the-art techniques. All source code has been made publicly available via GitHub [19] in order to facilitate future advances in this research domain.

References

1. Alieyan, K., Almomani, A., Anbar, M., Alauthman, M., Abdullah, R., Gupta, B.B.: Dns rule-based schema to botnet detection. Enterprise Information Systems **15**(4), 545–564 (2021)
2. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J.A., Invernizzi, L., Kallitsis, M., et al.: Understanding the mirai botnet. In: 26th {USENIX} security symposium ({USENIX} Security 17). pp. 1093–1110 (2017)
3. Barradas, D., Santos, N., Rodrigues, L., Signorello, S., Ramos, F.M., Madeira, A.: Flowlens: Enabling efficient flow classification for ml-based network security applications. In: Proceedings of the 28th Network and Distributed System Security Symposium (San Diego, CA, USA) (2021)
4. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al.: P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review **44**(3), 87–95 (2014)
5. CAIDA: (Jun 2021), `http://www.caida.org/data/passive/passive_dataset.xml`
6. of Canada, P.: Bill c-28, `https://www.parl.ca/DocumentViewer/en/40-3/bill/C-28/third-reading`
7. Cozzi, E., Vervier, P.A., Dell'Amico, M., Shen, Y., Bilge, L., Balzarotti, D.: The tangled genealogy of iot malware. In: Annual Computer Security Applications Conference. pp. 1–16 (2020)
8. Dainotti, A., King, A., Claffy, K., Papale, F., Pescapé, A.: Analysis of a "/0" stealth scan from a botnet. IEEE/ACM Transactions on Networking **23**(2), 341–354 (2014)
9. Fachkha, C., Bou-Harb, E., Keliris, A., Memon, N.D., Ahamad, M.: Internet-scale probing of cps: Inference, characterization and orchestration analysis. In: NDSS (2017)
10. Garcia, S., Grill, M., Stiborek, J., Zunino, A.: An empirical comparison of botnet detection methods. computers & security **45**, 100–123 (2014)
11. Guerra-Manzanares, A., Medina-Galindo, J., Bahsi, H., Nõmm, S.: Medbiot: Generation of an iot botnet dataset in a medium-sized iot network. In: ICISSP. pp. 207–218 (2020)
12. Hauser, F., Häberle, M., Merling, D., Lindner, S., Gurevich, V., Zeiger, F., Frank, R., Menth, M.: A survey on data plane programming with p4: Fundamentals, advances, and applied research. arXiv preprint arXiv:2101.10632 (2021)
13. Intel: Intel® tofino™ 3 intelligent fabric processor brief, `https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-3-brief.html`
14. Jepsen, T., Alvarez, D., Foster, N., Kim, C., Lee, J., Moshref, M., Soulé, R.: Fast string searching on pisa. In: Proceedings of the 2019 ACM Symposium on SDN Research. pp. 21–28 (2019)
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Kumar, A., Lim, T.J.: Edima: Early detection of iot malware network activity using machine learning techniques. In: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT). pp. 289–294. IEEE (2019)

17. Letteri, I., Della Penna, G., De Gasperis, G.: Botnet detection in software defined networks by deep learning techniques. In: International Symposium on Cyberspace Safety and Security. pp. 49–62. Springer (2018)
18. McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based clustering. Journal of Open Source Software **2**(11), 205 (2017)
19. NetSecResearch: (Sep 2021), `https://github.com/NetSecResearch/INC`
20. Networks, E.: Programmable Tofino switches for data centers, `https://www.edge-core.com/productsInfo.php?id=335`
21. Pastore, M., Calcagnì, A.: Measuring distribution similarities between samples: A distribution-free overlapping index. Frontiers in psychology **10**, 1089 (2019)
22. Pour, M.S., Mangino, A., Friday, K., Rathbun, M., Bou-Harb, E., Iqbal, F., Samtani, S., Crichigno, J., Ghani, N.: On data-driven curation, learning, and analysis for inferring evolving internet-of-things (iot) botnets in the wild. Computers & Security **91**, 101707 (2020)
23. Rahbarinia, B., Perdisci, R., Lanzi, A., Li, K.: Peerrush: Mining for unwanted p2p traffic. Journal of Information Security and Applications **19**(3), 194–208 (2014)
24. Sandbox, T.: (Aug 2022), `https://hatching.io/triage/`
25. Sanvito, D., Siracusano, G., Bifulco, R.: Can the network be the ai accelerator? In: Proceedings of the 2018 Morning Workshop on In-Network Computing. pp. 20–25 (2018)
26. Sapio, et al.: Scaling distributed machine learning with in-network aggregation. arXiv preprint arXiv:1903.06701 (2019)
27. Sapio, A., Abdelaziz, I., Aldilaijan, A., Canini, M., Kalnis, P.: In-network computation is a dumb idea whose time has come. In: Proceedings of the 16th ACM Workshop on Hot Topics in Networks. pp. 150–156 (2017)
28. Schubert, E., Sander, J., Ester, M., Kriegel, H.P., Xu, X.: Dbscan revisited, revisited: why and how you should (still) use dbscan. ACM Transactions on Database Systems (TODS) **42**(3), 1–21 (2017)
29. Siracusano, G., Bifulco, R.: In-network neural networks. arXiv preprint arXiv:1801.05731 (2018)
30. Stratosphere: Stratosphere laboratory datasets (2015), retrieved March 13, 2020, from `https://www.stratosphereips.org/datasets-overview`
31. Tanabe, R., Tamai, T., Fujita, A., Isawa, R., Yoshioka, K., Matsumoto, T., Gañán, C., Van Eeten, M.: Disposable botnets: examining the anatomy of iot botnet infrastructure. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–10 (2020)
32. Turkovic, B., Kuipers, F., van Adrichem, N., Langendoen, K.: Fast network congestion detection and avoidance using p4. In: Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies. pp. 45–51 (2018)
33. Xiong, Z., Zilberman, N.: Do switches dream of machine learning? toward in-network classification. In: Proceedings of the 18th ACM Workshop on Hot Topics in Networks. pp. 25–33 (2019)
34. Xu, Z., Chen, L., Gu, G., Kruegel, C.: Peerpress: Utilizing enemies' p2p strength against them. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 581–592 (2012)
35. Yang, F., Wang, Z., Ma, X., Yuan, G., An, X.: Switchagg: A further step towards in-network computation. arXiv preprint arXiv:1904.04024 (2019)
36. Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., Luo, X.: Detecting stealthy p2p botnets using statistical traffic fingerprints. In: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). pp. 121–132. IEEE (2011)

37. Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., Wu, J.: Poseidon: Mitigating volumetric ddos attacks with programmable switches. In: the 27th Network and Distributed System Security Symposium (NDSS 2020) (2020)