PAutoBotCatcher: A Blockchain-based Privacy-preserving Botnet Detector for Internet of Things*

Ahmed Lekssays*, Luca Landa, Barbara Carminati and Elena Ferrari

DiSTA, University of Insubria, Varese, Italy

ARTICLE INFO

Keywords:
Botnet Detection
Blockchain
Privacy
Internet of Things
Security

Abstract

Botnets have become a major threat in the Internet of Things (IoT) landscape, due to the damages that these sets of compromised IoT devices may cause. To increase their attacks' success, modern botnets are designed in a distributed manner, following a P2P structure. Recently, several botnet detection solutions have been proposed. Among them, community behavior analysis solutions seem to be promising because of their high detection accuracy. However, such solutions are not optimized for real life scenarios since they only run in a static mode, that is, reading all network traffic at once. As such, they do not support real-time data analysis. In order to handle such issue, these solutions should run in a dynamic distributed environment where different actors participate in the detection process. However, this collaborative environment brings up the issue of trust among the actors.

To address this issue, in this paper, we present *PAutoBotCatcher*, a dynamic botnet detection framework based on community behavior analysis among peers managed by different actors. *PAutoBotCatcher* leverages on blockchain to ensure immutability and transparency among all actors. To optimize continuous detection while keeping good accuracy, we design a set of optimization techniques, such as caching detection's output and pre-processing the shared network traffic. In addition, we leverage on different privacy-preserving techniques to protect devices from re-identification during the botnet detection process. We have extensively tested our solution to show its effectiveness and to demonstrate that blockchain is a good solution for dynamic botnet detection.

1. Introduction

IoT devices are having a drastic growth since the last 10 years because of their usefulness in many industrial applications. They have increased from 13.4 billion in 2015 to 38.5 billion in 2020 with a percentage of 285%.[1] Due to this growth, IoT devices have become an attractive target for attackers to perform various attacks, such as DDoS (Distributed Denial of Service). For example, Dyn, a major DNS Provider, faced one of the largest known DDoS attacks performed using compromised IoT devices and able to reach a bandwidth of 1.2 Tbps [1].

IoT devices are considered the weakest link in companies' security chain since they are not usually well tested and secured against cyber attacks due to, for example, the adoption of weak passwords and unencrypted network services.[2] In addition, they have low computation power to run sophisticated security solutions. As a result, attackers can easily inject malicious software (malware) in IoT devices to take control over them or steal private information [33].

Among the various kinds of attacks threatening IoT devices, botnets are used heavily in leading DDoS attacks. A botnet is a set of compromised devices that are controlled by a botmaster using Command and Control (C&C) servers. Botnets consist of three main parts: bots, a Command and Control (C&C) server, and a botmaster [7]. Traditional botnets are designed in a centralized way, where the control is done through a single C&C server that can be a single point of failure. However, modern botnets are designed in a decentralized way, where each bot is a C&C server, so it becomes challenging to detect and stop these botnets, since control can be done from various devices [7].

In the last few years, many botnet detection algorithms have been proposed, the most promising are those based on community behavior analysis [8]. These algorithms assume communication among bots as a distinct feature of distributed botnets. In other words, these algorithms exploit the fact that each bot plays the C&C server role, and this makes communications among such bots higher than the ones of benign devices. However, in real settings, these solutions might not be so effective as they have to cope with botnet attacks running on different realms (e.g., sub-networks), each one controlled by a different actor (e.g., different IoT vendors, ISP providers, etc.), which does not give the complete picture of the real network flow. In order to make this solution effective, the detection process should run across several realms, assuming that involved actors trust each other and share their data to collaboratively detect malware. However, in a real life scenario, the solution should be deployed in large networks, which makes the trust component much challenging to achieve. Since we cannot assume different actors trust each other, there is the need of a framework that facilitates

*This work has received funding from the Marie Sklodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. Additionally, it has been partially supported by CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

*Corresponding author

alekssays@uninsubria.it (A. Lekssays)

ORCID(s): 0000-0001-7511-2910 (A. Lekssays); 0000-0001-7511-2910 (B. Carminati); 0000-0001-7511-2910 (E. Ferrari)

[1] https://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020

[2] https://www.enisa.europa.eu/publications/baseline-security-recommendations-for-iot

the process of sharing malware detection related data.

AutoBotCatcher has been proposed as a preliminary solution to cope with trust issue in data sharing for malware detection [32]. It leverages on the PeerHunter community detection algorithm [47] and exploits a permissioned blockchain to dynamically and collaboratively detect botnets. In AutoBotCatcher, gateways monitor their sub-networks and collect the network flows locally. Then, they locally convert network flows into a PeerHunter-supported format and send them to the blockchain. Using the network flows, blockchain peers construct a communication graph via smart contract execution. Then, this graph is used to detect communities and thus botnets. To ensure the correct execution of the detection, AutoBotCatcher detection process leverages on smart contracts whose execution has to be validated via blockchain distributed consensus, in particular via Practical Byzantine fault tolerance (PBFT) consensus algorithm.

The work presented in [32] illustrates only the theoretical design of AutoBotCatcher. Deploying it in a real life environment comes with various challenges. First, botnet detection in IoT environments is particularly challenging because detection process needs to be fast and accurate, but IoT devices have limited resources. There is, therefore, the need to optimize the botnet detection algorithm to run dynamically in an incremental manner. Second, in AutoBotCatcher, devices' IP addresses are shared in a plain-text source-destination format in the blockchain. However, since blockchain is transparent, the sharing of this information might represent a violation of the privacy of devices' owners. Indeed, the detection process relies on constructing a graph from the source-destination entries. This graph might reveal the structure of gateways' sub-networks including their devices and links they make with other devices. Thus, there is the need of complementing AutoBotCatcher with a privacy-preserving layer able to avoid devices' re-identification by, at the same time, supporting a good accuracy and fast botnet detection.

In order to face the above-mentioned challenges, in this paper, we introduce *PAutoBotCatcher* which is a blockchain-based privacy-preserving IoT botnet detection solution. We leverage on blockchain to ensure a correct execution of the detection process without assuming trust among the different involved actors (e.g. IoT devices' vendors, internet service providers). Each actor is represented by an equal number of peers that run the detection process individually and agree on the final output in order to take the necessary actions after detection, like cutting off the infected devices from the network (see Section 3). To optimize AutoBotCatcher, we have introduced a technique to incrementally cache the updates of AutoBotCatcher's detection results in order to improve the running time while maintaining the detection's accuracy (see Section 5). Moreover, *PAutoBotCatcher* adopts a set of privacy preserving strategies, like graph anonymization and IP address randomization, to avoid the re-identification of IoT devices during the botnet detection process.

Contributions. The main contributions of this paper can be, therefore, summarized as follows:

- optimizing and modifying the community botnet detection algorithm used by AutoBotCatcher (aka PeerHunter) to run dynamically in an incremental manner;
- designing a privacy-preserving layer that can support dynamic graph construction for botnet detection;
- implementing the first privacy-preserving blockchain-based botnet detector for IoT;
- running extensive tests on PAutoBotCatcher by simulating a real life situation to demonstrate that blockchain is a feasible solution for real-time botnet detection.

Outline. The remaining of this paper is organized as follows. Section 2 gives an overview of the main concepts behind AutoBotCatcher [32], like PeerHunter algorithm, blockchain, and Hyperledger Fabric[3] blockchain framework. After that, Section 3 presents AutoBotCatcher building blocks. The privacy-preserving layer is illustrated in Section 4, whereas Section 5 presents the implementation details of PAutoBotCatcher. Then, security and privacy issues are discussed in Section 6. Section 7 discusses the results of our extensive tests, whereas Section 8 introduces related work. Finally, Section 9 concludes the paper and discusses future work.

2. Background

In this section, we provide preliminary information on the PeerHunter algorithm, blockchain, and Hyperledger Fabric, whereas Table 1 reports the main symbols used in the paper.

2.1. PeerHunter

PeerHunter [47] is a community behavior analysis approach capable of detecting botnets that communicate in a peer-to-peer manner. It has been demonstrated to be accurate in detecting bots of unknown types without the need of a knowledge base.

PeerHunter uses mutual contacts as the main feature to cluster bots into communities. Since P2P botnets communicate with each other, there is a higher probability to share a mutual contact compared to P2P legitimate hosts [42].

In PeerHunter, a mutual contacts graph is an undirected weighted graph $MCG = (V, E)$, where V is a set of vertices representing hosts and $E \subseteq V \times V$ is a set of edges, where $(u, v, w) \in E$ means that vertex u and vertex v share w mutual contacts.

Figure 1 shows an example of MCG for six nodes, where the edge between each pair of nodes denotes the number of their mutual contacts. For instance, node 2 and 4 share two contacts. In PeerHunter, mutual contacts graphs are represented as a 2D weighted adjacency matrix, referred to as *Mutual Contacts Matrix (MCM)*, where $MCM_{u,v}$ denotes the number of mutual contacts between u and v.

In its original formulation [47], PeerHunter has been designed as a static algorithm able to process a single dataset of

[3] https://www.hyperledger.org/projects/fabric

Table 1
Table of symbols

Symbol	Description
Γ	A detection round
Δ	Duration of a detection round
Δ_t	Starting timestamp of a round
MCG	Mutual contacts graph
MCM	Mutual contacts matrix
MCR	Mutual contacts ratio
DD	Diversity density
AVGDDR	Average diversity density ratio
AVGMCR	Average mutual contacts ratio
NT	Network flows transaction
α	Network flows threshold in a round
MCG_{Δ_t}	Mutual contacts graph at round starting at Δ_t
MCM_{Δ_t}	Mutual contacts matrix at round starting at Δ_t
$MCM_{u,v}$	Mutual contacts between nodes u and v

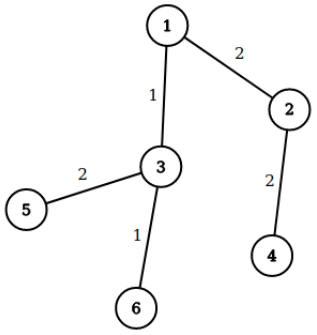

Figure 1: An example of Mutual Contacts Graph.

network flows at once. It exploits two numerical parameters to decide if a host belongs to a botnet or not. The first metric is the average of the diversity density computed for all hosts in network (*AVGDDR*). Given a node u, its diversity density (*DD*) is the number of distinct 16-bit IP prefixes in u's network flows' destination IP addresses. The second metric is the mutual contacts ratio (*MCR*) between a pair of hosts. Given two nodes, *MCR* is the number of mutual contacts between them, divided by the number of total distinct contacts that they have. PeerHunter considers the average of the mutual contacts ratio *AVGMCR*, that is, the average of *MCR* of all hosts. The thresholds for *AVGDDR* and *AVGMCR* are pre-defined at the deployment of the system, so when a host exceeds them, it is labeled as a bot.

2.2. Blockchain

Blockchain is a distributed ledger, maintained by a peer-to-peer network, that stores transactions modeling the results of interactions between peers [39]. Transactions are stored into a structure formed as a chained set of blocks, aka the blockchain. Not all the transactions submitted by peers are added to the blockchain, but only those that are considered valid according to the reference domain. Rules that need to be checked to assess the validity of a transaction are encoded into smart contracts, which are autonomously-executed programs.

Blockchain is designed based on the assumption that there is no universal trust among all participants [45]. It relies on the concept of distributed consensus, so a block (i.e., transactions contained in the block) is considered valid if the majority of the peers agree on its validity. This step is referred to as reaching consensus. Blockchain has different consensus protocols (e.g., Proof of Work (PoW), Proof of Stake (PoS), Byzantine Fault Tolerant (BFT) variants) [10].

In our paper, we will focus on Practical Byzantine Fault Tolerant (PBFT) consensus which is a variant of BFT, that operates even in the presence of some malicious nodes participating in the consensus process. It is based on the leader-follower paradigm so that a leader can sequentially order transactions to be processed by the nodes participating in consensus. So, they will validate all transactions in the same order. The leader is elected among nodes participating in the network (aka orderers), thus the "leader" role is not permanent (see Section 2.3 for more details). It is worth mentioning that PBFT is designed to properly operate if malicious nodes are strictly less than $\frac{1}{3}$ of total nodes.

The properties of blockchain being a distributed ledger makes the blockchain immutable, so it is a tamper-resistant technology. All the participating members in the blockchain have a copy of the ledger, so if a malicious actor wants to tamper with the blocks that were already validated, it needs to modify all the versions stored at the level of any member, which is impossible unless the malicious actor compromises a certain number of the participating members, which depends on the adopted consensus algorithm.

There are two types of blockchains: public and permissioned blockchains. Public blockchains are accessible by everyone and any peer can read and write to the blockchain. In other words, all the peers can participate in the consensus process. For permissioned blockchains, only selected and predefined peers can participate in consensus. So, only the predefined peers can write to the blockchain and access the shared data [22].

In PAutoBotCatcher, we exploit a permissioned PBFT blockchain. We adopt it because we want limited membership. So, only selected organizations (e.g. IoT vendors, internet service providers, etc.) can join the network.

2.3. Hyperledger Fabric

We adopt Hyperledger Fabric as our permissioned blockchain framework. It represents a valuable choice to develop PAutoBotCatcher, as it is a performant and flexible blockchain framework. The main aspect that makes this framework a good choice is its scalability [22]. Hyperledger Fabric processes up to 3,000 transactions per second (tps) that can be scaled up to 20,000 tps with some pluggable modules [9]. In addition, Hyperledger Fabric supports different languages to implement smart contracts, such as Java, Golang, and JavaScript.

In what follows, we briefly describe the key concepts of this blockchain framework. We refer the interested readers to [12] for more details.

Peers. Peers are the network's members that can interact with the blockchain. All nodes can submit and read transactions. In addition, some nodes can also run smart contracts and participate in consensus.

Organizations. Organizations are entities that take part to the network.

World State. A world state or ledger state is a key-value database, where various entries (i.e., transactions' numbers in the transaction pool, transactions' statuses, current block, etc...) are saved.

Orderers. These are a subset of peers that aim to keep the ledger state consistent across the network. Orderers are responsible for ordering all transactions. They do not participate neither in the execution nor in the validation of transactions.

Endorsement policies. Each smart contract (chaincode in Hyperledger Fabric) has to be run by all peers that participate in consensus. Each chaincode might have its own endorsement policy stating how many peers have to execute it. For instance, a chaincode A might have an endorsement policy that states that only two peers must agree on the chaincode's output. On the other hand, there might be a chaincode B that requires all the peers to agree on its output (i.e., the default endorsement policy in Hyperledger Fabric).

3. AutoBotCatcher

AutoBotCatcher exploits a PBFT permissioned blockchain to dynamically and collaboratively detect botnets, based on PeerHunter community detection algorithm. AutoBotCatcher is meant to be deployed in environments where IoT networks consist of devices and gateways. In particular, devices could be independent IoT devices accessing internet (i.e., devices with cellular capabilities), or IoT devices behind NATs (Network Address Translation) connected to a gateway. In case of independent devices, they have their own unique IP address. In contrast, for devices behind NATs, the uniqueness of the IP address depends on the implementation of the NAT. Two main cases are possible: i) all devices in the sub-network of a router/gateway share the same prefix of the gateway's public IP address, but they differ in the suffix; or ii) all devices in the sub-network have the same IP address but each device is mapped to a specific port.

In AutoBotCatcher, gateways serve as interfaces between IoT devices from one side and blockchain and the internet from another side. They monitor the network flows in their sub-networks, store them locally, and use them to create a transaction, referred to as Network Transaction. These are, then, submitted to the blockchain for the validation.

In addition, the system includes particular peers, called *block generators*. These nodes run the botnet detection mechanism and participate in the coordination process to vote on the validity of the detection output. We assume that block generators peers are provided only by involved organizations (e.g., IoT vendors, ISPs, etc.) in equal shares.

AutoBotCatcher workflow is shown in Figure 2. The processes executed by block generators are represented as purple boxes which also denote the steps of the botnet detection algorithm. The cyan box represents the process executed by gateways, whereas the orange box represents a process that is executed by either gateways or block generators. The details of each step are discussed in what follows.

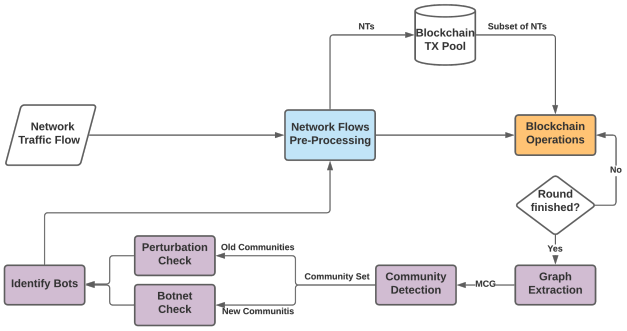

Figure 2: AutoBotCatcher workflow

Network flows pre-processing. This process is performed by gateways that constantly monitor and sniff network traffic in their sub-networks. Gateways keep track of IP addresses in *whitelists* and *blacklists*.

A blacklist contains the IP addresses of IoT devices that were identified as bots. In contrast, whitelists contain IP addresses of legitimate IoT devices. If a host was not detected as a bot in the previous execution but it is detected as bot in the current execution, it is automatically removed from the *whitelist* and moved into *blacklist*.

Definition 1. ***Network Flow Traffic.*** *([32]) Network flow traffic is modelled as a tuple:* $NetworkFlow = \langle ip_{src}, ip_{dst}, proto \rangle$*, where* ip_{src} *and* ip_{dst} *are the source and destination IP address of the sender and receiver IoT device, respectively, and proto is the adopted protocol, with* $proto \in \{tcp, udp\}$.[4]

Gateways create network flows transactions (NTs), formally defined as follows.

Definition 2. ***Network Flows Transaction.*** *([32]) A network flows transaction is defined as:* $NT = \langle Device_{adr}, NetworkFlow, Pool_{adr}, T \rangle$*, where* $Device_{adr}$ *is the unique public key of the gateway that sent the transaction,* $Pool_{adr}$ *is the public key of the transaction pool, whereas* T *is the timestamp when the transaction was submitted.*

Gateways are connected to a transaction pool, which is a space where transactions are stored before being validated by the network and finally added to the blockchain.

Blockchain operations. This step, shown in orange in Figure 2, is an incremental process executed in rounds at fixed time intervals. The process is executed by block generators that generate blocks of network flows transactions, NTs, received in the current round. To avoid flooding the system with too many blocks, we define a threshold α as the

[4]The considered protocols are *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP).

maximum number of NTs processed in a round. In order to generate blocks in a round Γ_{Δ_t}, a block generator is selected based on an election process.

The selected block generator takes a subset of NTs from the transactions pool containing all NTs generated during the last round, and not exceeding the α threshold, and forms a block. The block includes a *block header*, containing key-value pairs denoting the block number, and hashes of current and previous block. In addition, it contains *blockdata*, representing the submitted network flows transactions, and *block metadata*, containing the certificate and signature of the block creator, used to verify the block by network nodes. Then, it broadcasts the block to all the other block generators.

Since AutoBotCatcher exploits a PBFT blockchain, the selected block generator shall get the approval of at least $\frac{2}{3}$ of the total number of block generators to consider the block as valid.

To initiate the detection process and trigger the corresponding chaincode, gateways submit *detection transactions*. In particular, to avoid that two or more gateways submit detection transactions in the same time frame, AutoBotCatcher randomly selects the gateway that has to submit the detection transaction in a specific time frame.

Definition 3. ***Detection Transaction.*** *([32]) A detection transaction is defined as a tuple: $DT = \langle Device_{adr}, T_{start}, T_{end}, Pool_{addr}, T \rangle$, where T_{start} is the transaction's start timestamp, T_{end} is its end timestamp, $Device_{adr}$ is the unique public key of the device that sent the transaction, $Pool_{adr}$ is the public key of the transaction pool that the gateway is connected to send the transaction, and T is the transaction timestamp.*

Block generators run the detection process once a detection transaction is submitted. They run it on network flows transactions submitted in the timeframe specified by the detection transaction. The detection process starts with the graph extraction step explained in what follows.

Graph extraction. This is the first step of PeerHunter botnet detection algorithm. It is entirely executed by block generators that generate/update the mutual contacts graph MCG based on new NTs collected during round Γ_{Δ_t}. More specifically, block generators at round Γ_{Δ_t} perform the following steps on MCM_{Δ_t} to obtain $MCM_{\Delta_{t+1}}$: first, they update the number of mutual contacts, aka the weights of edges, based on network flows transactions collected in round Γ_{Δ_t}; second, they add vertices and edges representing new connections of IoT devices; finally, they remove vertices, if corresponding devices are no longer a part of the network.

Community detection. This is the second step of PeerHunter. This process is executed by block generators to perform dynamic community discovery, by taking as input the mutual contacts graph $MCG_{\Delta_{t+1}}$ updated in previous rounds and saved in blockchain state.

Perturbation check. This is the third step of PeerHunter botnet detection algorithm. This process is executed by block generators to keep track of the updates (i.e., IP addresses additions and removals) on the communities discovered in the previous round. For botnet communities, IP addresses are saved in two lists shared with gateways, namely *additions to blacklist* and *removals from blacklist*.

Botnet check. This is the forth step of PeerHunter and it is executed by block generators to classify communities as benign or botnet. The detection mechanism in PeerHunter is based on two observations: first, members of botnets communicate with each other to exchange commands, so, they have higher mutual contacts; second, botmasters or attack targets communicate with different nodes that are not a part of the botnet, called pivotal nodes [40], which lead to higher mutual contacts as well.

Identify Botnet. This is the last step of PeerHunter, which updates the local blacklists according to the results of the previous step. The process is executed by block generators by broadcasting the updated lists after confirming the validity of the updates.

4. Privacy-preserving Layer

The original version of AutoBotCatcher [32] does not protect the identity of IoT devices since network flows, shared in the blockchain, contain real IoT devices' IP addresses.

In general, from public IP addresses (of routers or IoT devices), sensitive information can be inferred, such as the geographic location, services that the IoT device is providing, IoT device's vendor, and thus its public vulnerabilities, etc. (see e.g., [18]).

Example 1. *Let us consider a network where there is a gateway/router which implements a NAT. Suppose the router has the public IP address 41.25.86.44, whereas an IoT device in its subnet has the local IP address 192.168.1.2. The IoT device uses port TCP/9100, and it is accessed from internet. The IoT device local IP address will be translated to the router's public IP address and port 41.25.86.44:9100. Thus, an attacker can infer the geographic location of this device, the service that this IoT device is providing (printing, in this case), the vendor (HP JetDirect printer, in this case), and also the public vulnerabilities of this printer, by checking exploits' databases, such as Exploit-db*[5].

A naive solution could be replacing IP addresses with fake ones. However, this is not enough, as an IoT device could be re-identified by its communication patterns, as the following example clarifies.

Example 2. *Let us consider the gateway's sub-network represented in Figure 3. where IoT devices' IP addresses have been randomized. Let us assume that an attacker knows that a target device in the sub-network communicates with exactly three other IoT devices. By exploiting this knowledge, the IoT device with randomized IP address 6.3.55.2 can be re-identified since it is the only node in the sub-network with three connections.*

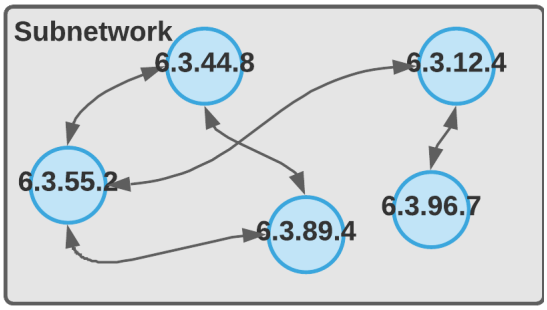

Figure 3: An example of identity leakage during network flows processing

Our privacy-preserving layer takes into account both scenarios where IoT devices have their own IP address (i.e., IoT devices with cellular capabilities), as well as the case where IoT devices are behind a NAT (i.e., connected to an access point). To avoid re-identification, we have adopted two anonymization techniques in our privacy-preserving layer. The first is the mapping of the real 16-bit prefix of an IP address to another valid but fake prefix, following the approach proposed in [3]. Moreover, to avoid devices' re-identification due to the knowledge of their connections, we anonymize the graph, leveraging on *(k,l)-anonymization* [6]. Among different graph anonymization techniques, such as k-anonymity [38] and l-diversity [24], we select (k,l)-anonymization since: i) it achieves the same privacy guarantees proposed by other techniques while running in polynomial time, which is critical due to the size of graphs we are dealing with; ii) it minimally adds edges to the generated graph, without removing edges or nodes, which preserve the accuracy of the detection algorithm. Figure 4 depicts PAutoBotCatcher's architecture. It is an extension of Figure 2 with additional anonymization modules. The first anonymization module is IP addresses' randomization executed by gateways, where they randomize IP addresses monitored in their sub-networks before publication in the blockchain. The second anonymization step is graph anonymization (shown in green). It is executed by the graph aggregator, an external trusted entity that does not participate in consensus, but it can read and send transactions. It gets the gateways' networks flows from the blockchain, aggregates them to form a merged graph, and anonymizes the resulting graph using *(k,l)-anonymization*.

4.1. IP Addresses Randomization

The first step of our privacy-preserving layer is to replace the IoT devices and gateways' IP addresses with fake IP addresses while keeping PeerHunter algorithm operational. We recall that the detection algorithm uses only IP addresses prefixes (i.e., the first 16 bits) to identify nodes and build a mutual contacts graph. Thus, this anonymization step should preserve the IP addresses' prefixes.

A first solution is that each gateway maps locally and independently the real IP addresses prefixes with fake ones producing its own mappings. However, in this way, communications among gateways are not preserved. Indeed, if gateway A and B talk to gateway C, A will use its own randomized prefix while B will use a different prefix. Thus, for the same gateway, two prefixes will be used. This property preserves the privacy of communications among gateways, but it limits the capabilities of PeerHunter in detecting botnets since the links between gateways will not be visible to PeerHunter. To avoid this situation, we assume that gateways share the assigned randomized prefixes. For example, if gateways A and B communicate with a gateway C, A and B will use the same prefix to communicate with C instead of using different prefixes for the same gateway. So, when a gateway is communicating with another one, it uses shared mappings instead of generating new ones. In order to keep the mappings available and consistent, they should not be saved locally by gateways since they have limited storage space, and they are not highly available. Hence, block generators maintain a distributed key-value store to save mappings of real prefixes with fake ones for each gateway. Each gateway encrypts its own mappings and stores them in the distributed key-value store. So, block generators do not have access to the mappings, but rather they just maintain the records. The shared mappings are encrypted with a key shared among gateways, so they can read and decrypt them (see Section 5 for implementation details). It is worth mentioning that external services' mappings are not shared with other gateways because they might reveal information about the services that an IoT device provides, its vendor, vendor's public vulnerabilities, etc. So, each gateway preserves its own external services' mappings.

The suffixes (i.e., the last 16 bits) are randomized at each round, so the same device can have multiple IP addresses. The randomization of the suffixes does not affect PeerHunter, and it adds a level of privacy where an attacker cannot infer the size of a gateway's sub-network (see Section 6.2). However, suffix randomization is not enough to protect an IoT device from re-identification since its communications are still exposed. Hence, the need for a graph anonymization technique arises.

4.2. (k,l)-Anonymization

In the context of PAutoBotCatcher, each gateway monitors its sub-network during a round Γ_{Δ_t} to submit its network flows in round $\Gamma_{\Delta_{t+1}}$. These network flows form a graph, referred to in what follows as sub-graph, for the sake of clarity.

The second step is anonymizing the graph obtained by merging the gateways' sub-graphs shared at each round. At each round, each gateway locally randomizes its IP addresses and encrypts its sub-graph with the public key of the graph aggregator. The encrypted sub-graph is shared in the blockchain (see Section 5.2 for implementation details). At the submission of a detection transaction, the graph aggregator gets the encrypted sub-graphs from the blockchain, decrypts them, and aggregates them to form an aggregated graph. We recall that the latter is a connected graph since we share the mappings of connections among gateways, so we preserve the links among the sub-graphs.

[5] https://www.exploit-db.com

In order to anonymize the aggregated graph, PAutoBotCatcher leverages on *(k,l)-anonymization* which aims at anonymizing graphs by introducing minimal additions to the overall graph structure [6]. *(k,l)- anonymization* assumes that an adversary can have the knowledge about the identifier of a node u, and the direct nodes connected to u. Therefore, for each node in the anonymized graph, *(k,l)- anonymization* requires that there should be at least k other nodes that share at least l of its neighbors. k and l depend on the degree of anonymization the user wants to achieve. The formal definition of *(k,l)- anonymization* is the following.

Definition 4. ***(k,l)-anonymized graph.*** *([6]) A graph $G = (V, E)$ is (k,l)-anonymous, if $\forall\ v \in V$ there exists a set of vertices $U \subseteq V$, where $v \notin U$ such that: 1) $|U| \geq k$, and 2) $\forall\ u \in U$, u and v share at least l neighbors.*

(k,l)- anonymization implies that for each node v in the anonymized graph, there are at least k other nodes sharing l of v's neighbors. (k,l)-anonymization attempts to hide the identity of nodes by creating groups of k nodes that look similar by sharing same neighbors, which are represented by the l factor. Any vertex in the original graph cannot be re-identified in one of its (k,l)-anonymous versions with a confidence greater than $\frac{1}{k}$. In order to anonymize a graph using *(k,l)-anonymization*, the work in [6] uses residual anonymity representing the level of anonymization of a graph.

Definition 5. ***Residual anonymity.*** *([6]) Let $G = (V, E)$ be a graph to be made (k,l)-anonymous. Let us consider a node $v \in V$ and suppose that k' other nodes in G share at least l of v's neighbors. We define the residual anonymity of v as $r(v) = max\{k' - k, 0\}$. The residual anonymity of G is $r(G) = \sum_{v \in V} r(v)$.*

The algorithm proposed in [6] computes the residual anonymity of each node and aggregates the residual anonymities such that the aggregation is equal to zero for an anonymized graph. It aims to add the minimum number of edges to get a (k,l)-anonymized graph. It runs in two steps: 1) choose randomly a node $w \in V$, add edges to form *(k+l)*-clique at w, and compute the residual anonymity of the graph G; 2) continuously find triplets of edges uv, vw, uw that maximally decrease the residual anonymity of the graph, add the new edges (i.e., uv, vw, and uw) to the graph, and update the residual anonymity of G. Finding triplets is a greedy process where the algorithm tries all edges combinations that maximally reduce the residual anonymity.

The performance of the (k,l)-anonymization algorithm [6] has been tested with different values of k and l for 4 real-life datasets of different sizes (see [26] for more details) . These experiments showed that (k,l)-anonymization can be used in real-life scenarios since it provides a good trade-off between information loss and performance.

Once the graph aggregator has anonymized the aggregated graph at $\Gamma_{\Delta_{t+1}}$, this is given as input to the PeerHunter algorithm to update the MCG used for botnet detection.

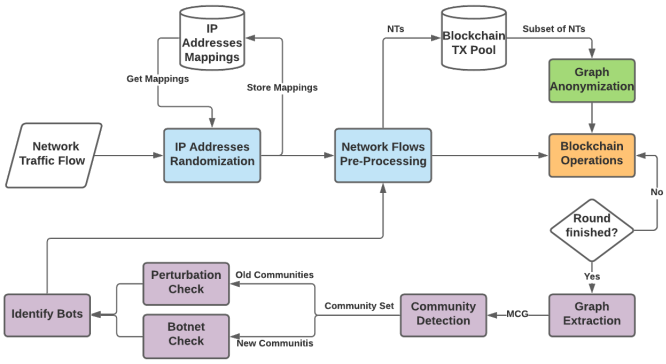

Figure 4: PAutoBotCatcher architecture

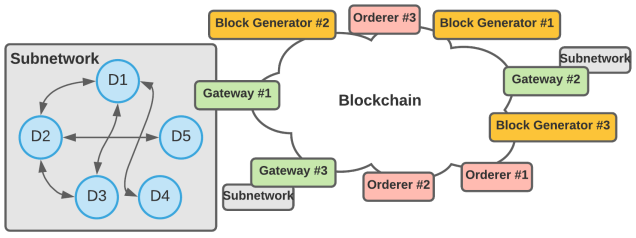

Figure 5: PAutoBotCatcher use case scenario

5. Implementation

In this section, we present the implementation of PAutoBotCatcher. In doing that, we base our discussion on the PAutoBotCatcher architecture illustrated in Figure 4.

5.1. Network Flows Uploading

Gateways handle network flows uploading. This process requires two steps (cyan boxes in Figure 4), namely IP addresses randomization and network flows pre-processing.

IP addresses randomization For each network flow, this step aims to randomize destination and source IP addresses by replacing the real IP addresses with fake ones. This process is done using a Python script implementing two main steps. The first is generating fake IP addresses' prefixes for IoT devices inside the gateway sub-network. To generate these fake prefixes, we randomly choose the first 8 bits from a valid range (from 0 to 255). The same process is used to generate the last 8 bits. Once the random values have been generated, we check if the obtained prefix is a valid IP prefix. We also check that the prefix has not been assigned to two devices in the same sub-network. This check cannot be performed on random prefixes created by other gateways, since mappings are encrypted (see later on in this section). However, the probability that two devices in different sub-networks have the same prefix is very small (this is estimated as the number of valid permutations that we can get from 16 bits prefix, that is 4.22×10^{-4}) and the mappings change in every round.

Mappings between the real IP addresses and the obtained randomized IP addresses are encrypted with the gateway public key and stored in Redis key-value store managed by block

generators. Thus, given a randomized IP address only the gateway who generates it is able to retrieve the real IP address by decrypting with its private key. This storage is useful to make the gateway able to check if the IP address have been already randomized, before generating a new fake IP address.

The second phase requires the randomization of IP address of devices outside the gateway sub-network, that is, external services, IoT devices or gateways with which internal IoT devices communicate. We recall that in case of gateways, their IP addresses have to be randomized with the same fake IP prefix through all sub-networks to preserve sub-networks' connections in the aggregate graph. For this purpose, prefixes assigned to gateways have to be known by all the gateways. To support this sharing, the mapping between gateway's real IP address and randomized one is encrypted with a key shared among all gateways and stored on the Redis store. As such, when the gateway has to randomize the IP address of another gateway, it checks on Redis store if the IP address has already been assigned a randomized IP address. If not, it will generate a new one.

Network pre-processing Once IP addresses have been randomized, network flows have to be packed in a network flows transaction (see Definition 2). This step is done by the gateway, which also saves the flows in a compressed record on the ledger state. The key used for this record consists of the timestamp of record generation and the gateway's public key (to identify records generated on the same timestamps), in such a way to speed up time range queries.

5.2. Graph Anonymization

As introduced in Section 4, the submission of network flows is done in rounds. More precisely, after the local IP randomization, each gateway encrypts its sub-graph with the graph aggregator's public key and share it in the blockchain. Then, the graph aggregator gets the sub-graphs from the blockchain, decrypts them in order to aggregate and anonymize them. Graph anonymization is implemented by two main components: the mapper and the builder. The mapper is an utility that maps IP addresses to integers for an easier nodes representation. The builder anonymizes the graph using *(k,l)-anonymization*, by adding a minimal number of edges. The added edges are converted to the IP format and appended to the anonymized network flows that will be submitted in the next round. Once the aggregated graph is anonymized, the graph aggregator publishes it in the blockchain for botnet detection. Since constructing an anonymized graph is a heavy process, the implementation was done in C++, and all these components are run sequentially.

5.3. Blockchain Operations

In this section, we describe how we have implemented the blockchain operations (orange box in Figure 4). Before this, we need to introduce the notion of detection state.

5.3.1. Detection State

The botnet detection process has to be done in a incremental way, to avoid re-running the process from scratch every time new network flows are submitted. For this purpose, we need to store any progress after each execution of the botnet detection process. In particular, we need to store the adjacency list of each node in the network, the latest status of the mutual contacts graph MCG and the updated list of detected bots. These information are stored via *Detection State*, defined as follows.

Definition 6. ***Detection State.*** *Let V be the set of nodes in the anonymized graph. Let $AdjLists = \{AdjList(v) \mid \forall v \in V\}$ be the set of all adjacency lists, denoted as $Adj\ List(v)$, for each v in V. Let $MCG = (V, E)$ be the current mutual contacts graph built on the anonymized graph, and $blacklist$ be the list of detected bots so far. A detection state DS is a tuple $\langle AdjLists, MCG, blacklist\rangle$.*

According to Definition 6, *DetectionState* has three parts: *AdjLists*, *MCG* and *blacklist*. The state contains a different key-value for each distinct adjacency list, where the host is the key and the set of connected hosts are the value. At the end of the detection round, *AdjLists* is updated only if any change happened from the previous round. The other two elements of *DetectionState* do not occupy relevant memory space (considering that the *MCG* is compressed), so they are directly written on the ledger at the end of each round.

Finally, the *MCG*, the *blacklist*, and the hashes of key-value pairs storing each distinct adjacency list are concatenated together and hashed into a unique value. These hash values are saved in a dedicated state *DetectionStateMetadata*, used to record the last version of the *DetectionState* and containing the metadata to properly retrieve it when needed.

Example 3. *Let us consider the code snippets shown in Figure 6. It considers a network with two nodes* 78.77.09.12 *and* 66.56.45.99 *connected to some external nodes. It shows an example of two versions of the detection state. An initial version containing the adjacency lists of all nodes in the original empty graph, an initialized Mutual Contacts Graph, and an empty blacklist. The second version contains the updated adjacency lists, the updated Mutual Contacts Graph, and the updated blacklist, which contains the detected bot* 23.67.45.23.

5.3.2. Consensus

A basic requirement of our system is the agreement on the botnet detection output, which is achieved thanks to the distributed consensus. We recall that the consensus is required for every transaction (i.e., network flows and detection transactions in our scenario) and is executed according the adopted endorsement policies (see Section 2). In PAutoBotCatcher, we designed two different endorsement policies for network flows transactions and detection transactions.

Regarding network flows transactions, these are submitted only by gateways which do not participate in consensus. As such, we do not require additional validation rules for their consensus, by adopting the default endorsement policy which requires validation from any organization. We recall

Figure 6: Examples of detection states

in PAutoBotCatcher, Hyperledger Fabric runs in a permissioned setting, thus each organization is registered. This allows network flows uploading without adding useless overhead to other organization peers. On the other hand, botnet detection transactions require consensus from all organizations. We recall that we assume that each organization participates with the same number of block generators to ensure the fairness of the consensus process. As such, the endorsement policy for the *DetectionState* transactions requires that all organizations express their validation on them. Thus, the *DetectionState* is stored in all block generators. After that, if consensus is not reached, the transaction is rejected, and it will not be broadcasted in the blockchain.

5.4. Botnet Detection Algorithm

Botnet detection consists of the purple boxes in Figure 4. We recall that PAutoBotCatcher leverages on PeerHunter for botnet detection (see Section 2.1). The original PeerHunter algorithm takes as input the whole dataset (aka network flows) and extracts the bots based on detected communities. However, to cope with the dynamic nature of our scenario, we made some changes to the original PeerHunter so that it can support detection on incremental network flows. In particular, we have re-designed PeerHunter to accept a *DetectionState* as input (see Definition 6).

The detection process is initiated by gateways when they submit a detection transaction. We recall that the detection process is performed by block generators. As depicted in Figure 4, the first phase is the extraction of edges for each host from the anonymized graph shared by the graph aggregator. The extracted edges are used to update the mutual contacts graph (*MCG*) that will be included in the round's *DetectionState*.

Given that the final purpose of PeerHunter is to detect hosts belonging to a P2P botnet, the original PeerHunter implementation builds the mutual contacts graph by adding only those hosts having a behavior similar to P2P hosts. To select these hosts, PeerHunter checks different conditions on hosts connections (e.g., destination diversity of each host), which are evaluated on the whole dataset [47]. However, in our scenario, since we process the dataset incrementally, it is required to check the conditions incrementally. As such, we have to track which conditions are satisfied over time and how their states change by edges/nodes updates. For this purpose, in the *MCG*, we add edges regardless of PeerHunter original conditions, so that we can keep track of every pair of hosts that has at least a mutual contact. To maintain PeerHunter accuracy, the only edges that will be used in the community extraction phase are the ones complying with PeerHunter requirements for *MCG* creation. By keeping track of all mutual contacts, our PeerHunter implementation can easily determine in every round which existing edges should be considered again for communities extraction.

6. Security and Privacy Analysis

In this section, we present an analysis of the privacy and security guarantees of PAutoBotCatcher.

6.1. Threat Model

In this section, we introduce the assumptions on the main entities involved in PAutoBotCatcher, namely IoT devices, gateways, block generators, and the graph aggregator. Many of them are commonly adopted by several proposals targeting the IoT scenario (e.g., [14, 34, 35]).

Block generators. We assume that each organization (e.g., IoT vendor, internet service providers, etc.) that actively participates in PAutoBotCatcher is represented by an equal number of peers serving as block generators. We do not assume that block generators are trusted as they could behave maliciously to bring benefits to their corresponding organization, such as invalidating other organizations' transactions. However, we rely on blockchain to ensure the correct smart contract execution, via distributed consensus. Since in PAutoBotCatcher we adopted the PBFT consensus protocol, we assume that $\frac{2}{3}$ of block generators are trusted (see Section 6.3 for more details). It is worth mentioning that this requirement can be reduced to $\frac{1}{2}$ with less efficient distributed consensus protocols, like Proof of Work.

Gateways. Gateways interact with the blockchain by submitting and reading transactions, but they do not participate in consensus. We assume that gateways are honest-but-curious. They are honest in the sense that they follow the protocol, but they are curious about the identities, communication patterns, and network topologies of other gateways.

IoT devices. We assume them as malicious, because they can be easily compromised. Notice that IoT devices do not interact with the blockchain.

Graph aggregator. We assume that it is a trusted external entity. It is a part of the blockchain that can read and send transactions, but cannot participate in consensus.

6.2. Privacy Attacks

PAutoBotCatcher aims to protect devices' identities and communication patterns. These data are collected by gateways. In particular, according to the threat model explained

above, each gateway has to: i) submit only legitimate network flows observed in its sub-network, ii) change the real IP addresses to fake IP addresses in the corresponding network flows, and (iii) encrypt their sub-graphs before publication in the blockchain. Privacy issues that may happen in PAutoBotCatcher under the assumption of honest-but-curious gateways are mainly related to: *1) leaking real IP addresses*; *2) re-identifying an IoT device based on its connections with other devices*; *3) inferring external services with which a gateway is communicating*; *4) inferring the belonging of an IoT device to a gateway*; *5) inferring gateway's sub-network topology*, and *6) inferring the type (i.e, IoT device or gateway) of a communicating device*.

Leaking real IP addresses. We recall that IP addresses are mapped to fake prefixes (i.e., first 16 bits) and randomized suffixes (i.e., last 16 bits) at the level of the gateway. Then, they are shared in the blockchain as Network Flows Transactions. Thus, the only entity that knows the real IP address of a device is its gateway.

Re-identifying an IoT device based on its connections with other devices. We recall that, according to the proposed solution, the locally built sub-graphs by each gateway are encrypted before being shared in the blockchain. The local sub-graphs are then aggregated to form an aggregated graph. The latter is anonymized by the graph aggregator. We leverage on (k,l)-anonymization privacy guarantees [6] which ensure that a node cannot be re-identified with a confidence of more than $\frac{1}{k}$.

Let us consider a curious gateway A that wants to know the links of a gateway B in the aggregated graph. We consider that gateway A communicated previously with gateway B. The curious gateway A knows its original and anonymized sub-graphs. In addition, it can identify its node and the gateway B node with which it connected to in the anonymized aggregated graph. So, a curious gateway A can infer: i) the fake links added to its node in the aggregated graph, and ii) the number of links that the node it connected to (i.e., a node in gateway B sub-network) has in the aggregated graph. However, it cannot tell if the links are real or fake (except from its own links).

Inferring external services with which a gateway is communicating. Since each gateway has its own mapping of external services with which it communicates, a curious gateway cannot infer the service it provides. For example, let us consider two gateways A and B whose anonymized IP addresses are 65.24.58.11 and 158.36.89.14, respectively. Suppose the two gateways communicate with the same weather forecast service. Since, for the same service, the prefixes used by the two gateways are different, so the external service cannot be inferred from the network flows.

Inferring the belonging of an IoT device to a gateway. Since the same IoT device can have different anonymized random IP addresses, a curious gateway cannot infer another gateway's network topology (e.g., the size of network). IoT devices and the gateway itself have the same prefix, but different randomized suffixes at each round. For example, let us consider the worst case where a gateway A has only one device, and suppose that the real external IP address of the gateway is 78.52.52.3. Suppose that the IoT device communicates with anonymized IP addresses 58.36.9.3 and 47.25.31 .56, respectively. The first communication from the IoT device in gateway A to the first anonymized IP address is represented in the graph as 96.33.25.14-58.36.9.3 where the second communication is represented as 96.33.44.78-47.25.31 .56. We can notice that the prefix 96.33 is the same, since it is the mapping of 78.52. But the suffix is randomized (25.14 in the first communication and 44.78 in the second one).

Inferring gateway's sub-network topology. As we mentioned previously, IoT devices and gateways can have multiple anonymized IP addresses, so a curious gateway cannot infer a target gateway's network topology (e.g., the exact number of devices in a target gateway's sub-network).

Inferring the type (i.e, IoT device or gateway) of a communicating device. Since IoT devices use the same prefix for both internal and external communications (e.g., 78.52 is mapped to 96.33), the type of the communicating device cannot be inferred from its anonymized IP address.

6.3. Security Attacks

Blockchains are vulnerable to some security attacks that have been deeply discussed in the literature [21, 20, 27]. In what follows, we consider majority attacks as the most relevant kind of attacks for PAutoBotCatcher.

Majority attacks. Majority attacks aim to control the system by having a majority of entities that participate in consensus. Controlling the system allows the majority to invalidate transactions, include only transactions that it wants in a block, etc. Blockchains have different percentages to achieve majority depending on their consensus algorithm. For example, for PoW and PoS, the majority is reached after 51%. In contrast, for PBFT, the consensus protocol adopted by PAutoBotCatcher, the majority can be reached by only taking down $\frac{1}{3}$ of the system [28]. In PAutoBotCatcher, each organization is represented by an equal number of block generators, so an organization cannot have an higher number of block generators than other organizations to control the system. Since PAutoBotCatcher leverages on PBFT, it assumes $\frac{2}{3}$ of the block generators to be trusted, so more than $\frac{1}{3}$ of organizations should collude to take down the system.

We are aware of other attacks that threat blockchain and P2P solutions in general, such as collusion attacks, free riding, and DoS attacks. Collusion attacks refer to a kind of attack where entities in a distributed system collaborate to reach a common goal to interrupt the normal operation of the system. This attack can be performed by entities who participate in consensus (e.g., block generators) and entities who do not (e.g., gateways). In contrast, free riding is a type of attack where entities benefit from the system without contributing to it. In PAutoBotCatcher, gateways could read detection states shared in the blockchain without sharing their anonymized network flows in the blockchain. In addition, DoS attacks (Denial of Service attacks) refer to a type of attacks where individual participating entities in the blockchain try to flood the system with transactions to make

Table 2
Experiments settings

Environment	**CPU Cores**	8 vCPUs
	RAM	30 GB
	Hard Drive	128 GB SSD
	Operating System	Ubuntu 18.04 LTS (64-bit)
Platform	**Blockchain Framework**	Hyperledger Fabric 2.2
	Chaincode Language	Java
Dataset	**# of P2P Legitimate Hosts**	60
	# of P2P Botnet Hosts	37
	# of Network Flows	38,189,903
	Size	1.7 GB

the nodes participating in consensus unavailable. Thus, the services provided by the blockchain become unavailable as well. This attack can be also performed when participating entities collude to make the system unavailable, and it is known as Distributed Denial of Service (DDoS). However, since our threat model considers gateways to be honest but curious and $\frac{2}{3}$ of block generators to be trusted, these attacks are out of the scope for PAutoBotCatcher.

7. Experimental Results

In this section, we present results of experiments we carried out on PAutoBotCatcher. For this purpose, we consider the use case scenario shown in Figure 4, consisting of: 3 block generators, each one belonging to a different organization; and 3 gateways, representing 3 different sub-networks of IoT devices. We have chosen this architecture to simulate a real world use case, with three smart homes connected via a gateway and three vendors. We developed and tested our solution on one 64-bit Ubuntu 18.04 LTS Google Cloud Platform virtual machine with 8 vCPUs, 30 GB RAM, and 128 GB SSD. We built the permissioned blockchain using Hyperledger Fabric v2.2 with Java 8 chaincode. The IoT devices, gateways, and block generators were simulated as Docker[6] containers. The source code of our system is open sourced for reproduction. It is available in three main components: PAutoBotCatcher[7], Botnet Detection Smart Contract[8], and Privacy-preserving Layer[9]. Table 2 shows the settings used to conduct the experiments.

7.1. Datasets

To evaluate the accuracy and performance of PAutoBotCatcher, we need a dataset containing both P2P legitimate traffic as well as traffic with botnets. For this purpose, we built a dataset with three different types of traffic: (a) P2P legitimate traffic, (b) P2P botnet traffic, and (c) background network traffic.

[6]https://www.docker.com/
[7]https://github.com/Lekssays/pautobotcatcher
[8]https://gitlab.com/lucalanda/botnetdetectionchaincode
[9]https://github.com/Lekssays/networkflowanony

Table 3
P2P legitimate traffic (24hrs)

Application	# of Hosts	# of Flows	Size
eMule	16	3,581,559	134M
uTorrent	14	7,523,902	292M
Vuze	14	6,189,657	237M
FrostWire	16	3,287,939	125M

P2P legitimate traffic: we used the dataset in [47] and described in Table 3. The dataset contains traffic of well-known P2P applications. In particular, it contains traffic generated by 4 eMule hosts, 14 uTorrent hosts, 14 Vuze hosts, and 16 FrostWire hosts. We randomly chose 24 hours of network traffic for each application.

P2P botnet traffic: again, we used the dataset in [47] and described in Table 4, that highlights the number of infected hosts, number of network flows per bot, and the size of each file containing network flows. The dataset contains 13 hosts infected by Storm,[10] and 3 hosts infected by Waledac.[11] We randomly chose 24 hours of network traffic of each botnet.

Background network traffic: we used the dataset downloaded from MAWI Working Group Traffic Archiveday [11]. In particular, we downloaded 24 hours of network flows in the 2020/15/1 day, containing 92.4% of TCP traffic and 7.6% of UDP traffic, and a total of 133,824,832 anonymized network flows (see Table 5).

For our experiments, we have used ARGUS [2] on background network traffic dataset in order to convert the *.pcap* files to triple format (cfr. Definition 1). For the other two datasets, we have implemented a Python script to convert them to triple format.

We have mixed the three raw datasets in a way that respects a real life scenario and challenges the PeerHunter de-

[10]https://www.networkworld.com/article/2286172/storm–the-largest-botnet-in-the-world-.html
[11]https://threatpost.com/waledac-botnet-now-completely-crippled-experts-say-031610/73694/

Table 4
P2P botnet traffic (24hrs)

Botnet	# of Hosts	# of Flows	Size
Storm	13	8,603,399	312M
Waledec	3	1,109,508	41M
Kelihos	8	122,182	5M
ZeroAccess	8	709,299	26M
Sality	5	5,599,435	211M

Table 5
Background network traffic

Date	Duration	# of Flows	Size
2020/15/1	24 hrs	133,824,832	2.8G

tection process. In particular, we have considered that if we randomly mix the datasets, the probability that a host will not have any mutual contact is high, which makes the detection easier for PeerHunter. Indeed, less mutual contacts means a disconnected legitimate community which makes the detection algorithm easier. To avoid this, we mixed the datasets such that each internal host has at least one mutual contact. The resulting merged dataset contained 37 botnets hosts and 38,189,903 network flows.

7.2. Experiments

Since we did not change the PeerHunter or (k,l)-anonymization algorithms, we focused our experiments mainly on PAutoBotCatcher performance. Indeed, the only changes on PeerHunter/anonymization have been done to execute them as a dynamic algorithm. These algorithm changes have not affected the detection accuracy. This has been confirmed by an experiment, not included in the paper due to lack of space, where we compared the detection accuracy obtained by PAutoBotCatcher and PeerHunter when executed on the same dataset.

In our experiments, we adopt *(4,1)-anonymization*. This choice is motivated by the fact that (4,1)-anonymization ensures a reasonable trade-off between protection from re-identification and performance.

Regarding the network flow simulation, it is worth mentioning that Hyperledger Fabric can process *3,000 transaction per second (tps)*. It can also reach up to *20,000 tps* with some plug-and-play modules [9]. In our case, we used plain Hyperledger Fabric, with no additional modules. Thus, with 5 network flows in each transaction we simulated *15,000 network flows per second (nfps)*.

As shown in Figure 7, we have tested PAutoBotCatcher against the original version of AutoBotCatcher, theoretically introduced in [32], and the standalone implementation of PeerHunter. The tests were done with predefined thresholds for PeerHunter parameters, aka AVGDDR and AVGMCR (see Section 2.1). For AutoBotCatcher, the processed flows per peer are 2,500 nfps. Thus, with the setting of 3 peers and 2 threads per peer, the system handled 15,000 nfps with

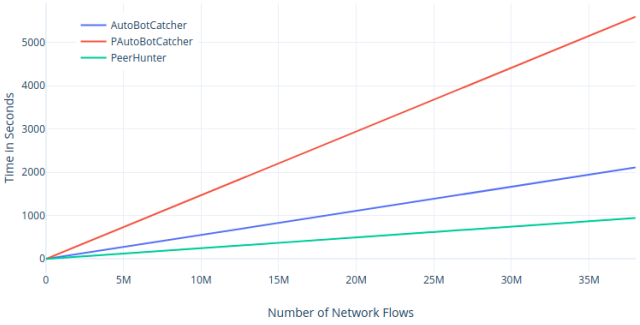

Figure 7: PAutoBotCatcher, AutoBotCatcher and PeerHunter execution time

no issue. We recall that also AutoBotCatcher dynamically runs the detection process based on rounds, where a round, with this configuration, contains on average 650,000 network flows. AutoBotCatcher was able to process 38,189,903 network flows in 2,122 seconds on average.

To have a fair comparison with standalone PeerHunter, we divided the dataset into chunks of 650,000 network flows to simulate rounds also in PeerHunter. Experiments shown that PeerHunter was able to process the same number of network flows of AutoBotCatcher but in 954 seconds. Based on this, we can estimate an overhead of 1,168 seconds due to the blockchain architecture. In addition, PAutoBotCatcher has the anomymization layer. As represented in Figure 7, PAutoBotCatcher took more time. In particular, it takes 5,396 seconds to process the same number of network flows. Our experiments show that our privacy-preserving layer processes a network flow in 8.5736×10^{-5} seconds on average.

Our experiments show that both AutoBotCatcher and PAutoBotCatcher can be used in a real life scenario, because both solutions process a network flow in 5.5564×10^{-5} seconds and 1.413×10^{-4} seconds, respectively. This processing time includes also the latency that occurs when Hyperledger Fabric is flooded by transactions. This flooding is due to the fact that our settings include only 3 hosts, simulating 37 infected hosts with more than 7 million transactions in 2h. With a broader network, this latency can be reduced. This let us suggest that PAutoBotCatcher can be a good solution for real-time data processing use cases even under heavy load as we simulated.

8. Related Work

The field of P2P botnet detection has gone through extensive research due to the damage that botnets cause to businesses and organizations [17]. Researchers have been investigating the topic from various perspectives. Most of the proposed approaches can be categorized into two main groups: community behavior analysis and network traffic analysis. The main difference between the two is that to detect botnets, the latter exploits traffic information (e.g., packets size and contents, communication protocols). In contrast, community behavior analysis detects botnets based on their communication structure, analyzing the network and the links

between devices.

- *Community behavior analysis approaches.* Extensive research has been done in this area. For instance, authors in [5] have worked on identifying local members of P2P botnets using mutual contacts. In addition, authors in [23] have worked on performing group-level behavior analysis on network traffic using Support Vector Machine. The main drawback of these methods is that they assume a knowledge base that contains metadata of previously known botnets. Given that IoT ecosystems frequently evolve [4], unknown botnets should also be taken into consideration, so the approaches in [5] and [23] are impractical in an IoT context. To cope with this requirement, AutoBotCatcher exploits PeerHunter [47] that is able to detect unknown botnets which is a key factor in IoT environments, where new botnets emerge frequently.

- *Network traffic analysis approaches.* These approaches can be mainly categorized as: statistical approaches and feature-based approaches. Statistical approaches use mathematical methods to model botnet detection, whereas feature-based approaches are based on finding features that characterize botnets.

For statistical approaches, the authors in [44] have used Mahalanobis distance to calculate the correlation between nodes in order to detect botnets. Mahalanobis distance can express correlations among indirectly connected nodes based on their traffic with commonly connected nodes. In [44], an iterative algorithm was proposed to get the correlation coefficient between the nodes with a pre-defined threshold of 85% to detect P2P botnets. Authors in [19] have proposed a hybrid distributed solution that combines a quantitative model and distributed threat intelligence. The suggested method aims to strengthen collaborative threat intelligence. This method uses network flows with the advantages of quantitative analysis and rates quality credibility of community shared intelligence. However, this solution is not dynamic, and it performs heavy computation and analysis on the provided data, which makes it not suitable for IoT contexts.

For feature-based approaches, researchers focus on finding mutual features among different hosts that might form a botnet. Authors in [16] have worked on a machine learning model to detect traffic similarity. The authors proposed a two-layers approach for traffic classification based on non-P2P traffic filtering and conversation features in machine learning. The first layer filters the non-P2P traffic in order to reduce the traffic flows to be processed in the second layer where extraction of conversation features is done. Conversation features, in this layer, are based on data flow features and flow similarity. [36] proposed a model to track evolution of a botnet over time using Long Short-Term Memory (LSTM). The authors in [43] proposed a method to detect botnets based on extraction of the data packet size and symmetric intervals in flows, based on the concept of graphic symmetry combined with information entropy and session features. This combination allows obtaining features with better correlations that are used to detect botnets. Whereas, authors in [46] suggested a botnet detection system by identifying all hosts that are likely to be a part of P2P communications. Then, they derive statistical fingerprints to classify different types of P2P traffic. These fingerprints are finally used to distinguish P2P botnet traffic. Authors in [31] suggested an approach to detect botnets based on network traffic behaviors. The approach relies on detecting P2P bots before launching their attack. They evaluated five different machine learning techniques to satisfy botnet detection requirements: adaptability, novelty detection, and early detection. However, none of the suggested models satisfy all the requirements at once. Moreover, authors in [41] and [25] have worked on deep packet inspection (DPI) to analyze the content of network traffic, but this technique can be bypassed by encrypting C&C channels.

However, there are drawbacks in the aforementioned community behavior and network analysis techniques, that limit their usage in an IoT environment. From a performance perspective, IoT devices have low computational power, so it would be impractical to run machine learning models on them unless they are distributed which is not considered in the suggested models. From a usability perspective, the above described approaches can detect only botnets that have been known before and were included in the training dataset, which is impractical [42] because IoT environments frequently change [4].

PAutoBotCatcher exploits PeerHunter that supports detecting previously unknown botnets, which is an important requirement in an IoT context. In addition, it does not use any feature extraction method that can be tampered with by changing packet sizes, encrypting C&C channels, changing communication frequency, etc. However, it is a static method, so it is limited to the usage of an already existing dataset. PAutoBotCatcher receives network flows in real time and execute the detection process periodically without the need of re-initializing the process. So, it detects bots in real time and in an incremental way.

On the other hand, many work have exploited blockchain for botnet detection in IoT. The authors in [29] proposed a collaborative architecture that leverages on blockchain's smart contracts to mitigate DDoS attacks performed by vulnerable IoT devices based on IP addresses blacklisting. The authors in [13] suggested a defense mechanism against DDoS attacks performed by IoT devices using Ethereum blockchain. They integrated communication between IoT devices in an Ethereum instance, so attacks can be prevented using gas limit which is provided as a default feature of Ethereum. Furthermore, the work in [37] suggested a blockchain-based solution for collaborative detection of botnets performing DDoS attacks. It exploits smart contracts where rules to classify network traffic as malicious are defined, and snapshots of the network traffic, generated by IoT devices, are shared in the blockchain. After that, smart contracts are triggered to send alerts to system admins if consensus is reached on the type of traffic generated. Moreover, [30] proposed BIoSS, which is a blockchain-based framework for information sharing between deployed DDoS detection and mitigation mechanisms to create reports and check if they match previously defined network rules describing malicious be-

haviors. Furthermore, the work in [15] discussed a blockchain-based solution to build, distributed DNS services that are resilient to DDoS attacks, since this kind of attacks is usually led by botnets, and it causes internet unavailability for millions of users (see e.g., [1]).

PAutoBotCatcher differs from the aforementioned work in many levels. First, PAutoBotCatcher is a privacy-aware botnet detection solution unlike all the work discussed above. It protects the identities of hosts in the blockchain while keeping an accurate solution. Second, PAutoBotCatcher does not consider performing DDoS attacks as a detection criterion unlike the work discussed in [37], [30], and [15]. Rather, PAutoBotCatcher focuses on the C&C commands exchanged among peers instead of network flows of the ongoing DDoS attacks. This features allows PAutoBotCatcher to detect botnets before they cause damages to a system unlike the other discussed papers that detect botnets while they are performing the attacks. Third, PAutoBotCatcher focuses on P2P botnets which were not discussed in the aforementioned contributions. They focused only on centralized botnets while PAutoBotCatcher focuses on decentralized botnets. Finally, the work in [29], [30], and [15] did not discuss the scalability aspect of the proposed solutions even if they take IoT as a context. The work in [13] uses Ethereum, so the scalability issue is related to the blockchain framework itself. However, the work in [37] is similar to PAutoBotCatcher is this sense because the authors provided a scalability analysis of their solution. The authors in [37] exploit blockchain for information sharing of possibly DDoS network flows between all entities for independent analysis. So, each entity will judge if the network flows represent a DDoS attack or not. PAutoBotCatcher performs the same operation of sharing network flows but for all traffic. The work in [37] demonstrated that blockchain can be scalable for such network flows analysis. PAutoBotCatcher showed that detecting botnets from analyzing network flows in real-time using blockchain is feasible and scalable.

9. Conclusion and Future Work

This paper presented PAutoBotCatcher, a privacy preserving blockchain-based solution for IoT botnet detection. PAutoBotCatcher protects the identities of hosts by mapping their real identities to fake ones and introduces optimal changes to the network structure to protect devices from re-identification based on their links with other devices. In addition, PAutoBotCatcher runs in a real-time and incremental manner to ensure an optimized detection process by leveraging on caching each increment's results. We tested our system with a large dataset of almost 40 million network flows. We have maintained the same accuracy claimed by PeerHunter [47] which demonstrates that the minimal changes implied by (k,l)-anonymization did not affect accuracy. In addition, we have conducted performance experiments showing that our system can support more than 15,000 nfps and that blockchain adds an overhead of 1,168 seconds for the considered testing scenario. This result allows us to conclude that blockchain can be suitable for real life use cases where real-time data processing is needed.

PAutoBotCatcher can be extended in many different directions. First, we plan to improve the current version by designing a well developed synchronization system for detecting transactions' submission. In addition, we plan to exploit smart contracts to perform incidents' responses and mitigation in a systematic way. In addition, we plan to extend our system by implementing a malware containment mechanism that analyzes the scope of the infection and disconnects the infected devices automatically.

Acknowledgements

This work has received funding from the Marie Skłodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. Additionally, it has been partially supported by CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927. In addition, the authors would like to thank Dr. Anh-Tu Hoang and Mr. Ha Xuan Son for their helpful feedback.

Declarations

Funding. This work has received funding from the Marie Sklodowska-Curie Innovative Training Network Real-time Analytics for Internet of Sports (RAIS) supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 813162. Additionally, it has been partially supported by CONCORDIA, the Cybersecurity Competence Network supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

Conflicts of interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Availability of data and material. The authors declare that all the data used in experiments in this paper are freely available via the mentioned sources in the corresponding sections.

Code availability. The authors declare that they have made the source code of all the parts of PAutoBotCatcher: *PAutoBotCatcher Blockchain*[12], *Botnet Detection Smart Contract*[13], and *(k-l)-anonymization layer*[14] open source and they are available freely online.

CRediT authorship contribution statement

Ahmed Lekssays: Conceptualization, Methodology, Software, Writing - Original Draft. **Luca Landa:** Methodology,

[12] https://github.com/Lekssays/pautobotcatcher
[13] https://gitlab.com/lucalanda/botnetdetectionchaincode
[14] https://github.com/Lekssays/networkflowanony

Software. **Barbara Carminati:** Conceptualization, Methodology, Writing - Review & Editing, Supervision. **Elena Ferrari:** Conceptualization, Methodology, Writing - Review & Editing, Supervision.

References

[1] Antonakakis, M., et al., 2017. Understanding the mirai botnet, in: 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 1093–1110.

[2] Argus, 2019. Argus: Auditing network activity. https://openargus.org. Accessed: 2019-12-15.

[3] Backstrom, L., et al., 2007. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography, in: Proceedings of the 16th international conference on World Wide Web, pp. 181–190.

[4] Carminati, B., et al., 2016. Enhancing user control on personal data usage in internet of things ecosystems, in: 2016 IEEE International Conference on Services Computing (SCC), San Francisco, CA. pp. 291–298.

[5] Coskun, B., et al., 2010. Friends of an enemy: Identifying local members of peer-to-peer botnets using mutual contacts, in: Proceedings of the 26th Annual Computer Security Applications Conference, Association for Computing Machinery, New York, NY, USA. p. 131–140.

[6] Feder, T., et al., 2008. Anonymizing graphs. arXiv preprint arXiv:0810.5578 .

[7] Feily, M., et al., 2009. A survey of botnet and botnet detection, in: 2009 Third International Conference on Emerging Security Information, Systems and Technologies, IEEE. pp. 268–273.

[8] Gaonkar, S., Dessai, N.F., Costa, J., Borkar, A., Aswale, S., Shetgaonkar, P., 2020. A survey on botnet detection techniques, in: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), IEEE. pp. 1–6.

[9] Gorenflo, C., Lee, S., Golab, L., Keshav, S., 2019. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second, in: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE. pp. 455–463.

[10] Gorkhali, A., Li, L., Shrestha, A., 2020. Blockchain: a literature review. Journal of Management Analytics 7, 321–343.

[11] Group, M.W., 2020. Mawi working group traffic archive. http://mawi.wide.ad.jp/mawi/samplepoint-F/2020/202001151400.html. Accessed: 2020-01-20.

[12] Hyperledger, 2019. Hyperledger fabric: A blockchain platform for the enterprise. https://hyperledger-fabric.readthedocs.io/en/release-2.0/. Accessed: 2019-12-10.

[13] Javaid, U., Siang, A.K., Aman, M.N., Sikdar, B., 2018. Mitigating Iot device based ddos attacks using blockchain, in: Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems, pp. 71–76.

[14] Jiang, L., Tan, R., Lou, X., Lin, G., 2019. On lightweight privacy-preserving collaborative learning for internet-of-things objects, in: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 70–81.

[15] Karaarslan, E., Adiguzel, E., 2018. Blockchain based dns and pki solutions. IEEE Communications Standards Magazine 2, 52–57.

[16] Khan, R.U., et al., 2019. A hybrid technique to detect botnets, based on p2p traffic similarity, in: 2019 IEEE Cybersecurity and Cyberforensics Conference (CCC), Melbourne, Australia. pp. 136–142.

[17] Kolias, C., et al., 2017. Ddos in the iot: Mirai and other botnets. Computer 50, 80–84.

[18] Kumar, D., Shen, K., Case, B., Garg, D., Alperovich, G., Kuznetsov, D., Gupta, R., Durumeric, Z., 2019. All things considered: an analysis of iot devices on home networks, in: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 1169–1185.

[19] Li, J., et al., 2019. Distributed threat intelligence sharing system: A new sight of p2p botnet detection, in: 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia. pp. 1–6.

[20] Li, X., Jiang, P., Chen, T., Luo, X., Wen, Q., 2020. A survey on the security of blockchain systems. Future Generation Computer Systems 107, 841–853.

[21] Liu, Z., Luong, N.C., Wang, W., Niyato, D., Wang, P., Liang, Y.C., Kim, D.I., 2019. A survey on applications of game theory in blockchain. arXiv preprint arXiv:1902.10865 .

[22] Lu, Y., 2019. The blockchain: State-of-the-art and research challenges. Journal of Industrial Information Integration 15, 80–90.

[23] Lysenko, S., et al., 2019. Botgrabber: Svm-based self-adaptive system for the network resilience against the botnets' cyberattacks, in: International Conference on Computer Networks, Springer. pp. 127–143.

[24] Machanavajjhala, A., et al., 2007. l-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) 1, 3–es.

[25] Mizuno, S., et al., 2017. Botdetector: A robust and scalable approach toward detecting malware-infected devices, in: 2017 IEEE International Conference on Communications (ICC), pp. 1–7.

[26] Mortazavi, R., Erfani, S., 2020. Gram: An efficient (k, l) graph anonymization method. Expert Systems with Applications , 113454.

[27] Moubarak, J., Filiol, E., Chamoun, M., 2018. On blockchain security and relevant attacks, in: 2018 IEEE Middle East and North Africa Communications Conference (MENACOMM), IEEE. pp. 1–6.

[28] Nguyen, G.T., Kim, K., 2018. A survey about consensus algorithms used in blockchain. Journal of Information processing systems 14, 101–128.

[29] Rodrigues, B., Bocek, T., Lareida, A., Hausheer, D., Rafati, S., Stiller, B., 2017a. A blockchain-based architecture for collaborative ddos mitigation with smart contracts, in: IFIP International Conference on Autonomous Infrastructure, Management and Security, Springer, Cham. pp. 16–29.

[30] Rodrigues, B., Bocek, T., Stiller, B., 2017b. Enabling a cooperative, multi-domain ddos defense by a blockchain signaling system (bloss). Semantic Scholar .

[31] Saad, S., et al., 2011. Detecting p2p botnets through network behavior analysis and machine learning, in: 2011 Ninth Annual International Conference on Privacy, Security and Trust, Montreal, QC. pp. 174–180.

[32] Sagirlar, G., et al., 2018a. Autobotcatcher: Blockchain-based p2p botnet detection for the internet of things, in: 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA. pp. 1–8.

[33] Sagirlar, G., et al., 2018b. Decentralizing privacy enforcement for internet of things smart objects. Computer Networks 143, 112–125.

[34] Sharma, S., Chen, K., Sheth, A., 2018. Toward practical privacy-preserving analytics for iot and cloud-based healthcare systems. IEEE Internet Computing 22, 42–51.

[35] Shen, M., Tang, X., Zhu, L., Du, X., Guizani, M., 2019. Privacy-preserving support vector machine training over blockchain-based encrypted iot data in smart cities. IEEE Internet of Things Journal 6, 7702–7712.

[36] Sinha, K., et al., 2019. Tracking temporal evolution of network activity for botnet detection. arXiv preprint arXiv:1908.03443 .

[37] Spathoulas, G., Giachoudis, N., Damiris, G.P., Theodoridis, G., 2019. Collaborative blockchain-based detection of distributed denial of service attacks based on internet of things botnets. Future Internet 11, 226.

[38] Sweeney, L., 2002. k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10, 557–570.

[39] Tschorsch, F., et al., 2016. Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials 18, 2084–2123.

[40] Wang, J., et al., 2016. Botnet detection based on anomaly and community detection. IEEE Transactions on Control of Network Systems 4, 392–404.

[41] Wüstrich, L., 2016. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection .

[42] Yan, Q., et al., 2015. Peerclean: Unveiling peer-to-peer botnets through dynamic group behavior analysis, in: 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon. p. 316–324.
[43] Yang, Z., et al., 2019a. A feature extraction method for p2p botnet detection using graphic symmetry concept. Symmetry 11, 326.
[44] Yang, Z., et al., 2019b. P2p botnet detection based on nodes correlation by the mahalanobis distance. Information 10, 160.
[45] Zhang, C., Chen, Y., 2020. A review of research relevant to the emerging industry trends: Industry 4.0, iot, blockchain, and business analytics. Journal of Industrial Integration and Management 5, 165–180.
[46] Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., Luo, X., 2011. Detecting stealthy p2p botnets using statistical traffic fingerprints, in: 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN), IEEE. pp. 121–132.
[47] Zhuang, D., et al., 2017. Peerhunter: Detecting peer-to-peer botnets through community behavior analysis, in: 2017 IEEE Conference on Dependable and Secure Computing, Taipei. pp. 493–500.