Lightweight and Identifier-Oblivious Engine for Cryptocurrency Networking Anomaly Detection

Wenjun Fan, Hsiang-Jen Hong, Jinoh Kim, Simeon Wuthier, Makiya Nakashima, Xiaobo Zhou, Ching-Hua Chow, and Sang-Yoon Chang

Abstract—The distributed cryptocurrency networking is critical because the information delivered through it drives the mining consensus protocol and the rest of the operations. However, the cryptocurrency peer-to-peer (P2P) network remains vulnerable, and the existing security approaches are either ineffective or inefficient because of the permissionless requirement and the broadcasting overhead. We design and build a Lightweight and Identifier-Oblivious eNgine (LION) for the anomaly detection of the cryptocurrency networking. LION is not only effective in permissionless networking but is also lightweight and practical for the computation-intensive miners. We build LION for anomaly detection and use traffic analyses so that it minimally affects the mining rate and is substantially superior in its computational efficiency than the previous approaches based on machine learning. We implement a LION prototype on an active Bitcoin node to show that LION yields less than 1% of mining rate reduction subject to our prototype, in contrast to the state-of-the-art machine-learning approaches costing 12% or more depending on the algorithms subject to our prototype as well, while having detection accuracy of greater than 97% F1-score against the attack prototypes and real-world anomalies. LION therefore can be deployed on the existing miners without the need to introduce new entities in the cryptocurrency ecosystem.

Index Terms—Blockchain, Cryptocurrency, Bitcoin, P2P Network, Anomaly Detection, Statistical Analysis, Traffic Analysis

1 Introduction

Cryptocurrency builds on a distributed blockchain to forgo a centralized authority (e.g., a bank) in storing and processing the anonymous and censorless financial transactions. While public-key cryptography is used to make the transaction irrevocable (for keeping track of who owns how many coins and preventing repeated use of the same coin as in double-spending), the nodes are encouraged to use new accounts (i.e., hashed public keys) for each transaction for anonymity [1]. In other words, Bitcoin and other cryptocurrencies operate in permissionless environment where no control exists in trust and identity registration to join in the cryptocurrency operations. Such permissionless and identity-trustless requirements of cryptocurrency challenge the identifier-based mechanisms traditionally used in other applications, and affect the cryptocurrency designs including networking and the distributed consensus protocol.

Bitcoin's ingenious invention kicking off blockchain technology and R&D was the utilization of proof of work (PoW) for distributed consensus protocol to enable the distributed and permissionless operations for high-risk financial operations [1]. The PoW-based consensus protocol is built on competitions between participants (called *miners*) and is computationally fair, i.e., the greater the computational power, the more likely the miner is to find a *block* (corresponding to a valid PoW solution) and earn the financial reward. The computationally intensive nature of PoW makes the computing resources a critical priority for cryptocurrency operations, and therefore challenges the usage of a security mechanism with heavy computational overhead, because such cost is in direct competition with that for mining, which reduces the financial profit from the mining reward. Our work focuses on the cryptocurrencies based on PoW distributed consensus protocol due to its popularity. For example, as of May, 2021, the two most popular cryptocurrency implementations, Bitcoin and Ethereum[1], use PoW and those two cryptocurrencies alone account for more than 62% of the entire cryptocurrency implementations in market capitalization [2].

The distributed consensus protocol builds on the miners participating in the peer-to-peer (P2P) network to propagate the blocks/transactions to enable the synchronization of blocks and distribution of the new yet-to-be-processed transactions across the miner network. The P2P networking is critical because it provides the information and the inputs for the distributed consensus protocol and the rest of the blockchain operations. An attacker controlling such networking and information flow can breach the transaction integrity, e.g., double spending [3], [4], waste the mining effort [3], [5], [6], or jeopardize synchronization, e.g., by partitioning the network [4], [6]. Despite its critical importance, the P2P network for blockchain remains vulnerable owing to the permissionless requirement of cryptocurrencies and the broadcasting overhead over a large miner network. More specifically, the Bitcoin P2P networking messages are in cleartext and lack authenticity since the cost overhead of such mechanisms and the additional vulnerabilities make them prohibitive for the Bitcoin's broadcasting[2]. Due to the lack of protection, Bitcoin networking is vulnerable to attacks like Sybil, spoofing, DoS [8] and Eclipse [3]–[5]. To protect the vulnerable cryptocurrency P2P network by using anomaly detection, the approach needs to address two

1. Ethereum has had plans to transition to PoS, e.g., Ethereum 2.0 and Casper, but it is still in progress.

2. Bitcoin networking is already consuming a significant amount of networking resources for its transaction/block propagation, and signaling traffic [7], even without the additional overheads associated with the security mechanisms; the Bitcoin R&D community is working on addressing the challenge to make the networking more efficient, e.g., SegWit (decreasing the packet size) and sendcmp (greater efficiency in block traffic).

challenges: 1) cryptocurrency is permissionless that challenges the traditional identifier-based mechanism; 2) cryptocurrency is incentive-based and relies on computationally intensive mining. Though a number of anomaly detection approaches were proposed to secure the blockchain P2P network, they focused on identifying the cryptocurrency network entities and the user's behavior [9], [10]. Thus, they are often ineffective in the permissionless network environment. Also, they often use machine learning-based approaches [11]–[14] which unfortunately compete with the mining operations on the computational resources.

In this paper, we aim to secure the P2P network via detection while ensuring that the detection engine is practical to cryptocurrency nodes, tackling their unique challenges. To this end, we introduce the Lightweight and Identifier-Oblivious Engine (LION) that outperforms from the previous research and detection mechanisms to achieve the following two goals. First, LION is efficient in permissionless environment. It uses the networking information and avoids identifier-based analyses to defend against Sybil, spoofing, and Eclipse attacks, which are enabled in the cryptocurrency's permissionless environment because of the lack of identity- and registration-based trusts. Second, LION is lightweight so that we can practically deploy it on the existing miners as opposed to requiring separate entities for deployment. To achieve these design properties, LION is based on anomaly detection for modeling the normal networking behaviors to avoid modeling the ever-growing anomaly space in blockchain, and uses statistical analyses rather than machine learning to make it simple and lightweight in computations and deployable to the miners.

We take an empirical systems approach for performing the research. First, we prototype LION and incorporate it into an active Bitcoin node. Second, we test our scheme with real-world Bitcoin traffic, e.g., the milestone event of Bitcoin halving. Third, we prototype the networking threats on Bitcoin based on the current state of the art research in the field to further evaluate the scheme. LION is anomaly detection and therefore does not use the knowledge/signatures of the threats and can defend against other threats including those which are unknown or under-analyzed. Fourth, we include the system analysis and study how the scheme implementation affects the mining process and rate. Consequently, the experimental results show that LION yields less than 0.95% of mining rate reduction, in contrast to the state-of-the-art machine-learning approaches costing 12% or more depending on the algorithms, while having detection accuracy of greater than 97% F1-score against attack prototypes and real-world anomalies.

The contributions of this paper can be summarized as follows:

- We build a lightweight and Identifier-Oblivious Engine (LION) for anomaly detection in cryptocurrency blockchain P2P network addressing the unique challenges in cryptocurrency networking including those from the requirements of permissionless and computational resource efficiency.
- We analyze the LION design and study the design parameters (including the parameter choices, detection threshold, and detection time window size) and their impacts on the LION performance.
- We build a LION prototype on an active Bitcoin node and run the prototype on the real-world Bitcoin Mainnet to sense and collect the networking data. We use the prototype for evaluation and system analysis.
- We conduct a data-driven study and evaluation of LION. The evaluation includes separate attack prototypes attacking the LION-hosting node, and the testing of LION against the attack prototypes. We test LION not only using the attack prototypes but also against the real-world events without the attack prototypes.

The remainder of the paper is structured as follows: Section 2 describes the challenges, the motivation, and the threat model and scope. Section 3 proposes the design of LION including the architecture and the parameters and rules for detection. Section 4 shows the data collection and anomaly prototypes. Section 5 prototypes LION and analyzes the different parameters for detection. Section 6 presents the experimental results. Section 7 analyzes the LION's cost efficiency and mining impact. Section 8 presents the reactive peer connection rebuilding as an active countermeasure extending the anomaly detection. Section 9 presents the related work. Section 10 draws the conclusion to the paper.

2 Challenges, Motivations and Scope

This section describes the unique challenges in mining-based cryptocurrency and motivates the design of LION. More specifically, the cryptocurrency characteristics and requirements drive the LION to be anomaly-detection based, identifier-oblivious, and computationally efficient. Built on that, we also present the threat model and scope.

2.1 Cryptocurrency Networking Challenges

Cryptocurrency is permissionless challenging the identifier-based security models Cryptocurrencies are designed for the anonymity of the transactions to prevent regulations and censorship, for example, Bitcoin recommends using new keys/accounts for each transaction [1]. Such requirement to forgo centralized control and regulation makes the cryptocurrency operate in a distributed fashion with no explicit registration control, thus enforcing the *permissionless* nature of the cryptocurrency network. The participants instead generate their own identifiers and credentials[3]. Because the permissionless cryptocurrencies lack the trust in registration and identities (often provided by the trusted authority in other applications, e.g., web security), the traditional authentication or other identifier- or credential-based mechanisms are not viable to deter or prevent the threats

3. Blockchain applications to build decentralized applications beyond cryptocurrency applications, such as consortium blockchains, can operate in the permissioned environment with registration control. However, cryptocurrencies to generate new currencies and process financial transactions without a trusted bank generally operate in permissionless environment to avoid the centralization/regulation issues on the registration. There are exceptions even within the financial processing applications which uses permissioned blockchain, including Diem [15] which plans to implement a private cryptocurrency but does not exist yet, and BDB [16] which does not generate new currencies but just processes financial transactions using existing currencies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing

against source integrity, such as spoofing or Sybil. To demonstrate such threats, we prototype threats violating the source integrity in spoofing and Sybil attacks. Because the source integrity is not preserved, we design LION which is oblivious to the information provided by the identifiers but uses the networking traffic information instead.

Cryptocurrency is incentive-based and relies on computationally intensive mining Cryptocurrency relies on the participation of the distributed set of nodes, driven by the reward incentives. For example, double-spending becomes feasible if the cryptocurrency network does not have many nodes (and much computational power) participating in the PoW consensus protocol. Though cryptocurrency could use other lightweight consensus protocols for processing blocks, PoW is still the most widely used consensus protocol by the cryptocurrency P2P networks. Thus, in this paper, we focus on the PoW-based cryptocurrencies. Due to the absence of the centralized authority for governance, the incentives for implementing a security mechanism needs to be aligned with the interests of the subject node using the mechanism. Thus, we design LION which is lightweight and practical for the computation-intensive miners.

TABLE 1
Bitcoin message types with the numerical indices.

Index	Message Type	Notion
1	VERSION	HandShake Initiation
2	VERACK	Response to the VERSION
3	ADDR	Send a max of 1000 IP addresses
4	INV	Send a max of 50000 trans./blocks
5	GETDATA	Response to INV with a max of 50000 entries
6	GETHEADERS	Request up to 2000 headers after a given hash
7	TX	Response to GETDATA with a trans.
8	HEADERS	Response to GETHEADERS with trans. count
9	BLOCK	Response to GETDATA with non-compressed data
10	GETADDR	Request IPs from the peer's buckets of IPs
11	MEMPOOL	Request trans. from peer's mempool
12	PING	Request PONG
13	PONG	Response to PING
14	NOTFOUND	Response to GETDATA when has no that data
15	SENDHEADERS	Request headers
16	FEEFILTER	Ignores trans less than 8bytes
17	SENDCMPCT	Request a compact block
18	CMPCTBLOCK	Response to SENDCMPCT using short trans. IDs
19	GETBLOCKTXN	Request a BLOCKTXN message
20	BLOCKTXN	Response to GETBLOCKTXN (block and trans.)
21	REJECT	Inform the peer its sending message rejected
22	MERKLEBLOCK	Response to GETDATA using the inventory type
23	GETBLOCKS	Request up to 500 blocks after a given hash
24	FILTERLOAD	Set the filter that filters INV
25	FILTERADD	Add a bloom filter
26	FILTERCLEAR	Clear bloom filter

2.2 Motivations for LION

Because of the challenges aforementioned, LION is for anomaly detection and uses networking traffic information to analyze the behavior as opposed to using identifiers. It is also designed for efficiency so that it can be practically deployed on miners who prioritizes computational effort for mining and earning rewards.

Efficiency The efficiency and lightweight design is important to our design goal because we envision LION to be deployed on resource-frugal miners. LION is lightweight in its algorithmic implementation and execution by using anomaly detection, statistical analyses (as opposed to ML), and simpler rules and logic (e.g., limiting the number of parameters). We compare our approach with the prior ML-based approaches because of the popular use of ML in cryptocurrency networking security.

Anomaly detection *Anomaly detection* models the normal behavior as the reference profile to detect the anomaly if the event diverges from the normal reference profile, in contrast to the signatures or heuristics-based detection approaches modeling the abnormal/attack pattern or behavior. Anomaly detection is advantageous in both the simplicity in the reference modeling and the generality over the abnormal behaviors. The simplicity of LION makes it lightweight in overheads. Anomaly-based LION can also be applied for the unprecedented events, which lacks a prior knowledge of potential patterns and thus is difficult to establish the pattern/signature for the model in advance. Anomaly detection also provides greater resiliency against adversarial learning threats where the attacker tampers the training and the model. Furthermore, any message traffic anomalies should be concerning for bitcoin/cryptocurrency node. On the one hand, the network attacks incurred by the permissionless P2P network can be reflected by the message traffic state change. On the other hand, the message traffic state change involving anomalous block/transaction messages arrival will surely impact the Bitcoin node mining efficiency no matter what event is causing it.

Use networking traffic information We use the parameters measured from the networking traffic to drive our detection engine. The *networking traffic information* focuses on the frequency, timing, and size of the cryptocurrency application-layer messages, e.g., the Bitcoin's messages [17] are described in Table 1 with indices, as opposed to further inspecting the IP packets (e.g., distinguishing them by the source peers). We use the traffic information because our target application is in permissionless cryptocurrency, which makes spoofing or Sybil feasible. Against an attacker generating multiple fake identities in a permissionless environment lacking a prior trust, the identifier-based or packet-based approaches to distinguish the incoming networking arrivals to multiple distinct streams/sources and analyzing them based on such finer-granular streams can become ineffective. By stressing on the *identifier-oblivious* message traffic information rather than packet- or identifier-based information, LION is robust against such threats.

Incentives aligned to miners We implement LION on the existing miners who are also the beneficiaries of the detection and securing their networking connections. Because introducing separate entities implementing LION outside the miners can cause incentive issues (i.e., how to incentivize such security implementations especially while they may not benefit from the security implementation), we consider the LION implementation on miners[4]. LION should therefore be practical for the miners' deployment, providing constraints on the LION computational costs because the costs for the security mechanisms are in direct competition to those for mining. We therefore build LION to be computationally efficient and minimally intrusive of the miners' effort to mine and earn rewards.

4. Extending the LION implementation beyond the miner nodes, e.g., to build network-level security intelligence and analyses, is an interesting future work direction and is outside of scope for this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing

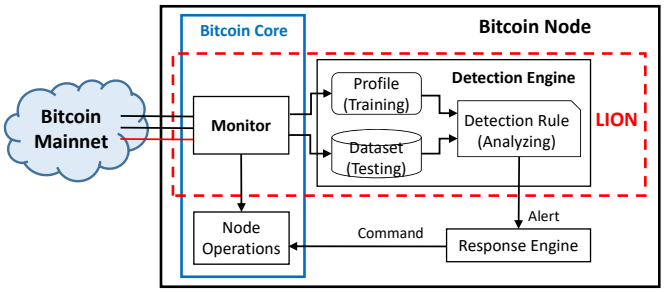


Fig. 1. An overview of the LION architecture.

2.3 Threat Model and Scope

Our work is applicable to a wide spectrum of threats because our detection engine is based on anomaly detection and focuses on the networking traffic on the victim. In fact, our detection can also be used against anomalies that are not caused by a malicious source. Because our work is not tied to a specific networking attack, the threat model includes any active networking node which can connect and inject networking communications to the victim node, e.g., supporting the Bitcoin Mainnet protocol and sending peer connection requests to the victim node. The permissionless aspect of the cryptocurrency P2P network, enabling the attacker to spoof or launch Sybil attack, makes the general threat model easy to realize for the attacker. Our threat model presents a low barrier and thus includes the threats requiring greater setup requirements for the active network adversaries, such as those requiring compromise in routing and networking service infrastructure [4], [6].

3 LION Design

In this section, we propose the design of the Lightweight and Identifier-Oblivious Engine (LION) to detect anomalies in permissionless cryptocurrency P2P network.

3.1 LION Architecture

This section describes the architecture of LION. Fig. 1 graphically presents an overview of the architecture, which contains two major components: Monitor and Detection Engine, while the architecture also includes several external connecting components which are Response Engine and Node Operations.

On the one hand, from a development perspective, Monitor and Node Operations are actually two functional modules of the Bitcoin Core application. In other words, they are already developed by Bitcoin Core, and we just make use of them when the Bitcoin node is running. By contrast, Detection Engine and Response Engine are the components developed by us from scratch. In particular, Detection Engine is designed to achieve the goal of detecting anomaly on the message traffic, and so it even emphasizes the internal components including Profile, Dataset, and Detection Rule. On the other hand, from a functionality perspective, this paper focuses on presenting Monitor and Detection Engine, since they mainly provide the functions for anomaly detection on the message traffic, while the subsequent use of the detection output for the active defense measure is achieved by Response Engine and Node Operations, which will be described in Section 8 as a complement to LION.

Further, we present the workflow of LION involving these corresponding components to accomplish the anomaly detection purposes, which includes three phases:

Phase I: Training Phase I stresses on training the model in order to create the normal reference profile for detection.

1) The Bitcoin node gets started to run and connects to the Bitcoin Mainnet, whereby Monitor can sense and collect all the arrival messages that include the traffic information like the message type, timestamp, byte size, clock cycle, etc.
2) Meanwhile, Dataset is used to store the data in a suitable format for the sake of being processed by other components.
3) Once enough data has been collected, Profile can train the model over the data to get the normal reference profile, which can be updated by training the model on the newly collected data.
4) The collected data stored by Dataset is mainly used for testing, in particular, the data observed during the run-time. Also, Dataset computes the thresholds of the model by comparing the training data and the testing data. We use three parameters to form the thresholds, which are described in Section 3.2.
5) Detection Rule includes the normal reference profile and the corresponding thresholds which formulate the detection rules for analyzing the run-time message traffic in order to perform anomaly detection. Thus, the outputs of Profile and Dataset are the inputs of Detection Rule.

Phase II: Detection Phase II focuses on using the LION's Detection Engine to perform anomaly detection on the message traffic.

1) Monitor keeps on capturing the message traffic and stores the data into Dataset.
2) After a certain timing window of data collection, Detection Rule starts to analyze the collected data within the window size and compare the results with the thresholds to determine whether the networking is anomaly or not.
3) If any anomaly is detected, Detection Engine will send an alert to Response Engine to notify it to take reaction against the anomaly.

Phase III: Response Phase III emphasizes the response after an anomaly is detected.

1) If Response Engine receives an alert from Detection Engine, it will process the alert and decide the reaction. Thereafter, Response Engine will send a command to Node Operations to inform the Bitcoin Core application to carry out the reaction.
2) Eventually, Node Operations will execute the command to perform the response in order to protect the Bitcoin node from the anomaly networking.

In addition to the workflow, we would like to clarify that LION can be deployed anywhere in the cryptocurrency P2P network (i.e., Bitcoin Mainnet in our case) as long as it has both the network sensor capability (represented by Monitor) for observing information transmission and the algorithmic computing capability (represented by Detection Rule) for analyzing the message traffic. Moreover, we focus

on the LION's implementation on the miner nodes because the miners both participate in the computation-intensive PoW consensus and relay the blocks and transactions via the P2P network. Therefore, in Fig. 1, all the components are inside the Bitcoin node. However, the design is not limited to deploying them into the miner node altogether as a monolithic system. They can be separated and placed in front of the miner node.

3.2 LION Parameters and Detection Rule

LION uses the Bitcoin's message traffic information and particularly the following parameters for the networking anomaly detection: the count rate n (the rate of message packet arrivals), the size rate s (bandwidth), and the relative frequency distribution Λ (distinguishing between the message types towards a Bitcoin node). These are standard parameters for traffic analyses, and the prototype evaluation in Section 6 provides high performances by using these parameters. There can be other useful parameters for detection, e.g., our dataset further includes timestamps for message arrival (such as instantaneous inter-arrival rates), clock cycles for processing received message, and peer connections, but we limit the number of parameters to make LION efficient and lightweight.

LION is based on the comparison of the current observation and the training-provided reference profile. For the volume-based parameters, i.e. the count-rate of arrivals (n) and the size-rate for networking bandwidth (s), there are ranges of networking volumes which are considered as normal. More specifically, $\tau_n = [\tau_{n,min}, \tau_{n,max}]$ and $\tau_s = [\tau_{s,min}, \tau_{s,max}]$ are the ranges for n and s, respectively. For example, if $n > \tau_{n,max}$, then the message packet counts of the arrival traffic is abnormally high and thus the anomaly is detected. For the relative frequency distribution across the message types (Λ), we use the correlation coefficient ρ to compare the observed/tested network traffic with the reference profile. The correlation magnitude ρ is between 0 and 1 where $\rho = 0$ indicates no correlation/similarity and $\rho = 1$ indicates that they are exactly identical, e.g., $\rho(\Lambda_{train}, \Lambda_{train}) = 1$. Hence, if such correlation between the testing and the training profile is less than the LION's frequency-distribution threshold τ_Λ, i.e., $\rho(\Lambda_{test}, \Lambda_{train}) < \tau_\Lambda$, then an anomaly is detected. LION detection rule aggregates the detection results from every parameter; such aggregation for the final detection output depends on the application requirement and the trade-off for the error types, as we investigate further in Section 5.

In addition, we define the time window size of the training dataset (T_{train}) and the time window size of the testing data (T_{test}). In other words, the detection engine makes the detection decision using the networking occurred in the past time window size of T_{test}. This also means that LION uses the entire entries within the timing window rather than per-entry to profile the features of the networking message traffic in order to perform the anomaly detection. All the variable notations are summarized in Table 2.

TABLE 2
Variable Notations.

Variable	Description	Section
n	Packet count rate over all messages (num/min)	§3.2, §5.1
s	Bandwidth size over all messages (Bytes/min)	§3.2, §5.2
Λ	Relative frequency distribution among messages	§3.2, §5.3
$\tau_{n,min}$	Threshold for the message counts (lower bound)	§3.2, §5.1
$\tau_{n,max}$	Threshold for the message counts (upper bound)	§3.2, §5.1
τ_n	Threshold of n, i.e., $[\tau_{n.min}, \tau_{n.max}]$	§3.2, §5.1
$\tau_{s,min}$	Threshold for the message size (lower bound)	§3.2, §5.2
$\tau_{s,max}$	Threshold for the message size (upper bound)	§3.2, §5.2
τ_s	Threshold of s, i.e., $[\tau_{s.min}, \tau_{s.max}]$	§3.2, §5.2
τ_Λ	Threshold for the relative frequency distributions	§3.2, §5.3
x	Range ratio for tuning τ_n	§6.1.1
τ_n'	The tuned range after x is applied to τ_n	§6.1.1
$\rho(\alpha, \beta)$	Correlation coefficient magnitude between vectors/distributions α and β	§3.2, §5.3
T_{train}	Time window size of the training data	§3.2, §5.4
T_{test}	Time window size of the testing data	§3.2, §5.4

TABLE 3
Machine Specification.

Operating system	Linux Mint 19.2 Tina (64-bit)
CPU	Intel Core i7 4GHz
Processor number	8
Memory	10 GB
Network Adapter	Intel PRO/1000 MT Desktop

4 Data Collection and Anomaly Prototypes

This section presents the Bitcoin node implementation first, and built on that, it describes the anomaly prototypes and the data collection. The networking data are collected from both the real-world Bitcoin Mainnet and the networking-anomaly prototypes (an attacker node introduced to the Mainnet-connected victim), informed by the research in Bitcoin P2P security. We train LION using the Normal data but control and configure the parameters and the detection thresholds using both the Normal data and the Abnormal data involving our attack prototyping. We then use LION to detect the real-world networking without our attacker prototype. This section provides the bases for the following sections, including the LION implementation prototype in Section 5 and the LION evaluation and analyses in Section 6.

4.1 Bitcoin Node Implementation

Our Bitcoin node implementation is based on Bitcoin Core (software version Satoshi 0.18.0 and protocol version 70015) with its default setting. We implement the Bitcoin nodes with equal specifications (see Table 3). In addition to the Bitcoin node hosting the LION engine, we implement other nodes for generating anomalies, and all the nodes have equal machine specifications.

Regarding the testbed setup, we implement multiple nodes and separately control each node's connections. We configure LION miner node's peer connectivity via random IP/peer selection while configuring the attacker node to only connect to our LION-hosting miner node. This enables us to better control the attack traffic so that any anomaly/attack traffic generated by our prototype is completely contained in our testbed and never spills over to the Bitcoin Mainnet to negatively impact the public Internet.

4.2 Networking Threat Prototypes

We prototype the networking anomalies by implementing the threats of denial of service (DoS) and Eclipse, which are actively being studied in cryptocurrency R&D. We also implement the more conventional threat vectors of spoofing and Sybil in Bitcoin because such attacks can nullify the identifier-based detection and used for DoS and Eclipse.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing

TABLE 4
LION Dataset Description (All the datasets are generated using the real-world Mainnet).

Dataset	Section	Description	Motivation	Training	Testing	Control	Attack Prototype
Normal	§4.3	Connected normally	Normal networking	○	○	○	
Abn-1p	§4.3	Only 1 peer connection alive	Eclipse attack		○	○	○
Abn-Syn	§4.3	Continuous block synchronization	Eclipse attack		○	○	○
Abn-DoS	§4.3	Under the DoS flooding attack	DoS threats		○	○	○
Bitcoin Halving	§4.4	Bitcoin reward halving on May 11, 2020 (occurs every 4 years)	Historical system-level change		○		
Public Node	§4.4	Additionally enable inbound connection requests	Miner node setup change		○		
Eclipse Attack	§4.4	Eclipse attack with random dropping of packets with 0.5 drop rate	Security (Eclipse attack)		○		○

4.2.1 Denial of Service (DoS)

The traffic flooding-based DoS corresponds to the repeated transmissions of the networking messages to overwhelm the receiving node's networking or computation resources. We use the Bitcoin `PING` message in our prototype even though any message can be used for flooding. `PING` is specific to the Bitcoin application and is different from the ICMP `ping` at the network-layer. Such flooding-based DoS changes the relative frequency across the messages because the flooded message type becomes the dominant (most frequent) networking message, which shifts the relative frequency distribution. The idea to detect such an anomaly is to find out the abnormal message frequency distribution compared by the normal frequency distribution. Further, to defend against the message flooding-based DoS attack, instead of using an external firewall to filter the traffic, the Bitcoin node can disconnect the peer connection directly.

4.2.2 Eclipse Attack

Eclipse attacker controls the peer connections to a victim node so that all the victim's connections go through the attacker [3]–[5]. When we talk about the Eclipse attack here, we focus on the Eclipse state/result instead of the specific Eclipse approach, since we use our prototype to make the Eclipse state. The attacker pretends to be multiple nodes using Sybil and manipulates the victim node's peer-table by populating it with its own IP addresses. Eclipse is powerful to have a range of devastating aftermath impacts, including the application-layer attacks of double spending and mining wastage. We focus on two threat vectors enabled by the Eclipse attack. The first threat is simply limiting the peer diversity of the victim node but relaying the blocks and transactions, and such threat yields unhealthy connectivity to the victim node, because the Bitcoin P2P network relies on the diversity of the peers, and the greater the independently-operating peers the victim node connects to, the more reliable block/transaction propagation it has. The second threat is to manipulate the block relaying for selfish mining [5], [18] or block withholding [19]–[21]. Such threat withholds the blocks but relays them sporadically and in groups and are devastating because they have the victim mine on outdated blocks wasting and undermining the victim node's mining effort. Our testing datasets described in Section 4.3 reflect these two networking threats and anomalies. If the Eclipse attack towards a Bitcoin node is detected, to make the node escape from the Eclipse state (i.e., the node is isolated from the rest of the Bitcoin network), all the current peer connections should be dropped immediately, and then new peer identifiers (which are not compromised by the Eclipse attacker) should be selected to re-establish peer connections in order to reach out to the rest of the Bitcoin network and synchronize the blockchain ledger.

4.2.3 Spoofing and Sybil Attack

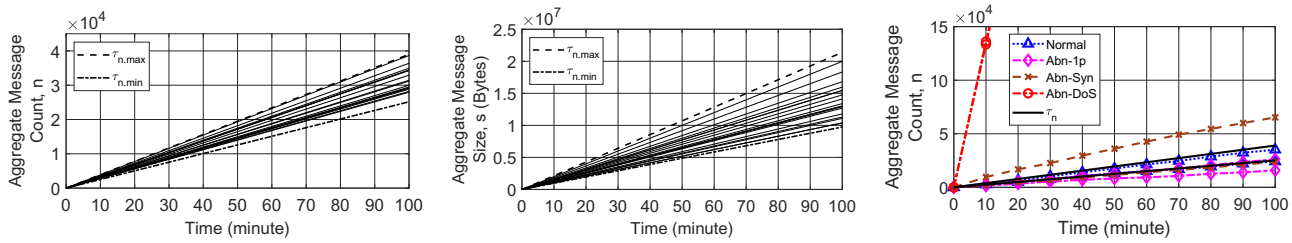
Bitcoin P2P network does not offer authenticity protection on the networking credentials. Thus, the threats based on false identifiers are effective. Spoofing corresponds to impersonating a known node (in this case, already connected to the receiving node hosting LION) and Sybil attack constructs an entirely new identifier (i.e., the IP address and the TCP port number). Because the Bitcoin P2P network uses the Transmission Control Protocol (TCP), spoofing requires knowing the TCP sequence number at the time of attack; the spoofing attacker needs to use the expected `seqnum` and `acknum` to inject the spoofed TCP segment into the target connection processed by victim node. Sybil attack, in contrast, does not even require such effort because the attacker can simply construct a new TCP connection by using a new IP address with a private TCP port. We implement both the Sybil and spoofing attacks as proof of concepts and use them to motivate LION. Spoofing and Sybil are quite general threats in permissionless networks, a direct way to defend against them is to provide cryptographic protection to connection, which however is very costly for a P2P network. Thus, an identifier-oblivious detection approach is proposed in this paper, and based on the detection results, the Bitcoin node can also choose to drop all the peer connections to resist against the spoofing and Sybil attacks.

4.3 Data for LION Training and Control

Our datasets are generated from the natural Bitcoin Mainnet activities through monitoring the Bitcoin messages on the public Internet. Table 4 describes the datasets used in this paper, including the typical adversary cases existing in the context of cryptocurrency.

We distinguish between the data for training and control so that the *training* of LION is for setting up the normal reference profile, which is a prerequisite for carrying out anomaly detection, and thus it only uses the Normal networking data. LION detects anomaly if the testing data diverges from the normal reference profile. On the other hand, the *control* involves the control and the configuration of the parameters and the detection threshold, to increase the detection performances. Because we control the attacking prototype and therefore know the ground truth of the data, we use the data involving the attack prototype for control in this case. After the LION's parameter control and configuration, we use it to test the real-world networking which does not involve our attacker prototype in Section 4.4.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing



(a) n (Training) (b) s (Training) (c) n (Testing and the range of τ_n)

Fig. 2. n and s training and testing.

The training for constructing the LION reference profile precedes the testing, and they occur separately in time. For training, the reference profile for LION is based on the networking monitoring of the real-world Bitcoin Mainnet, excluding the aforementioned attacker node that we control. Because the Bitcoin Mainnet is not free of attacking traffic, when collecting so-called normal networking date, we make the Bitcoin node proactively refresh all connections to randomly selected peers in every 50 minutes to prevent continuously connecting with misbehaving peers.

Thus, for the LION training and control, we monitor the LION node's connections for the Normal environment with no attack traffic (*Normal*). The Bitcoin `PING` message traffic flooding-based DoS threat provides the Abnormal-DoS dataset (*Abn-DoS*). Notice that the flooding traffic never exfiltrates to the Bitcoin Mainnet. The Eclipse attack prototype generates the other two abnormal cases. The Eclipse attacker simply limiting the peer diversity of the victim relaying the blocks/transactions yields the case of abnormal with only 1-peer connection (*Abn-1p*). Alternatively, the Eclipse attacker controlling and delaying the block relay timing for selfish mining and block withholding (as opposed to sharing the blocks as they arrive) yields the case of abnormally busy with continuous synchronization (*Abn-Syn*).

The ground truths of the above datasets are known (i.e. normal or abnormal) so that we use the Normal case to train the reference profile and use the abnormal cases to refine the detection thresholds for the LION engine. Our data collection includes the following information, which is richer/broader than what is required for the LION as is presented in this paper: the aggregate message count, summed message size in Bytes, clock cycles for processing the overall messages by the current interval; arrival message count, arrival message size and clock cycles during the current interval. Section 6 studies the importance of the parameter choices for the anomaly detection. The size of the dataset increases by 3.48 MB/hour, and we run our Bitcoin node to collect real-world data for more than 10 days.

For the testing results analyses in Section 5 and Section 6, we test 20 distinct datasets for each of the four testing scenarios: *Normal*, *Abn-DoS*, *Abn-1p*, and *Abn-Syn*. Each of the testing datasets is collected over the period of 10 block arrivals in expectation ($T_{test} = 100$ minutes since each block arrives in every 10 minutes on average by the PoW design) while the training dataset is collected over $T_{train} = 2,000$ minutes unless otherwise noted, e.g., Section 5.4 varies the timing window sizes for the training and testing to study their impacts.

4.4 Data for LION Run for Real-World Testing

In addition to the LION training and control datasets that we know the ground truths for controlling the LION engine, we sense and collect other datasets including those whose anomaly source is not controlled by our prototype, e.g., Bitcoin Halving and Public Node, which are also included in Table 4.

Bitcoin Halving refers to the historical system-level change of Bitcoin, which made the 12.5 BTC block reward go down to 6.25 BTC and occurred at block height 630000 and at 2020-05-11 13:23 in our Bitcoin prototype. Such change in the block reward is configured in the genesis block (the original block in the Bitcoin blockchain) and is scheduled to occur every four years. Due to the significance of such event since Bitcoin and cryptocurrencies are driven by such financial incentives, as the distributed miners uphold the integrity of the transactions because of such financial rewards, we study such case to investigate whether it has an effect on the networking.

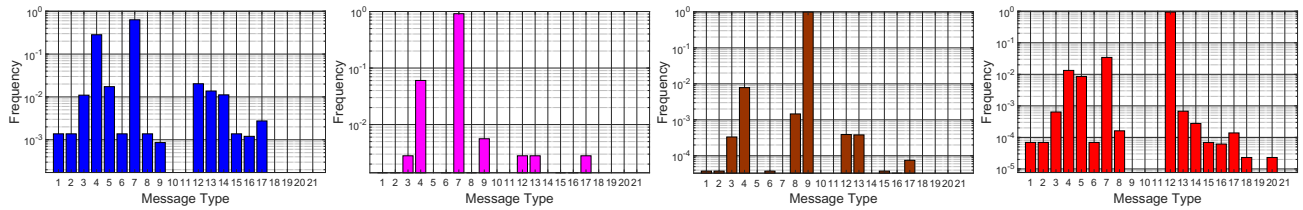
Public Node results from the change on the miner node setup, which enables/allows inbound connection requests. While the other datasets were connected to the nodes on the Bitcoin Mainnet, the connections were initiated by the LION node. In contrast, this Public Node dataset allows inbound connections and thus has greater fluctuations in the number of P2P connections. "Public Node" demonstrates a kind of event that the network setting of a node is changed by assigning a public IP address to it, whereby the node can be publicly reachable/routable. A private node (using a private IP address) has upto 10 outbound peer connections, while a public node can have more than that, in addition to outbound peer connections, it will have inbound peer connections. Normally, a public node would have more than 20 inbound peer connections, e.g., 22 [22], 24 [7].

Eclipse Attack builds on our attacker prototype (also used in Section 4.2) but the prototype behaves differently and simply drops the incoming relaying message packets from the Bitcoin Mainnet with a probability of 0.5. Such attack causes an inconsistency with the number of peers being tracked and the anomaly networking traffic. Section 6.2 presents the detection results involving the detection results when running LION against these scenarios.

5 LION Prototype and Analyses

This section provides the prototype- and experiment-based analyses of the different components of LION. We implement the LION prototype building on the Bitcoin node implementation. In this section, we focus on the per-testing analyses (except for the CDF in Fig. 4). The analyses emphasize the parameters of the message arrival rate (n), the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing



(a) *Normal* with ρ = 0.993 (b) *Abn-1p* with ρ = 0.946 (c) *Abn-Syn* with ρ = 0.074 (d) *Abn-DoS* with ρ = 0.02

Fig. 3. The sample frequency distributions Λ of the four Testing cases described in Table 4 and the corresponding ρ from the Analyzer. The testing datasets are color-coded, i.e., blue for *Normal*, magenta for *Abn-1p*, brown for *Abn-Syn*, red for *Abn-DoS*.

bandwidth (s), and the message frequency distribution (Λ), as described in Section 3.2. We divide the analysis of each of these parameters in to *Training*, *Analyzer*, and *Testing*. We also analyze the impacts of the timing window size for training and testing, which are T_{train} and T_{test} respectively.

5.1 Analysis of Packet Count Rate, n

Training Fig. 2(a) shows the n collected during training. More specifically, the n collected during training is divided into non-overlapping time window segments whose length is equal to $T_{test} = 100$ minutes. Because there are 20 such windows, Fig. 2(a) includes 20 curves corresponding to each segment samples in training.

Analyzer The training provides the maximum of the n samples, $\tau_{n,max}$ (the n corresponding to the greatest slope) and the minimum, $\tau_{n,min}$ (the n corresponding to the smallest slope), as highlighted in the legends in Fig. 2(a). We use $\tau_n = [\tau_{n,min}, \tau_{n,max}]$ to define the reference range as we discussed in Section 3.2. Based on the collected data for training, $\tau_{n,min} = 252$ messages per minute, $\tau_{n,max} = 390$ messages per minute, and $\tau_n = [252, 390]$.

Testing Fig. 2(c) shows the n for the four testing scenarios for control in Table 4. *Abn-DoS* provides the greatest message arrival rate because of the Bitcoin `PING` message traffic flooding to the LION node. *Abn-Syn* also provides abnormally large number of message arrival rate because the attacker causes continuous synchronization by withholding the previous blocks and submitting them altogether to the LION node. Fig. 2(c) additionally includes $\tau_n = [\tau_{n,min}, \tau_{n,max}] = [252, 390]$ (which has been derived from training depicted in Fig. 2(a)), and *Normal* dataset stays within τ_n. In contrast, *Abn-1p* shows an abnormally low number of message arrivals as there is only 1 peer connection.

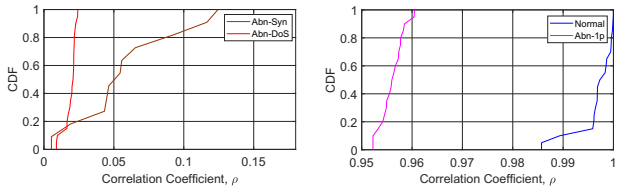
5.2 Analysis of Bandwidth Size, s

The analysis for bandwidth size, s is similar to the n analysis described in Section 5.1, and thus it is only briefly summarized in this section. Fig. 2(b) plots the s collected during training (*Training*), which includes the reference range of τ_s (*Analyzer*), and that in turn can be used in detecting anomalies which have the networking bandwidth outside of the normal reference range of τ_s (*Testing*).

5.3 Analysis of Frequency Distribution Across Messages, Λ

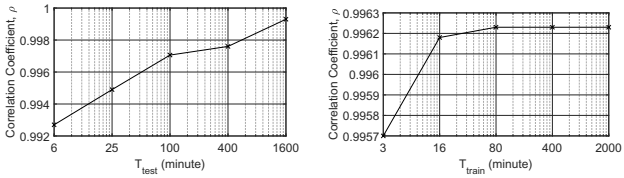
Training The training provides the message frequency distribution, and so the Λ of the reference profile is generated by the networking traffic from the Bitcoin Mainnet.

Testing The testing measures the normalized frequency



(a) ρ of Abn-DoS and Abn-Syn (b) ρ of Abn-1p and Normal

Fig. 4. CDF of ρ over 20 testing datasets.



(a) Varying T_{test} while fixing T_{train}=2000 minutes (b) Varying T_{train} while fixing T_{test}=100 minutes

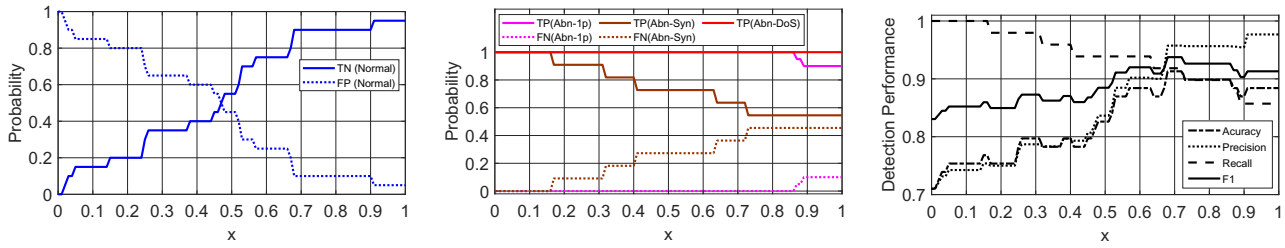
Fig. 5. The timing windows for testing/training vs. ρ when testing the Normal case.

distribution of the incoming message traffic across the testing datasets. Fig. 3 shows the sample frequency distributions, Λ for the four different testing cases described in Table 4. The horizontal axis indicates the different Bitcoin message types, while the vertical axis is the normalized frequency distribution and is plotted in logarithmic scale. It is worth mentioning that the message types that we focus on are from index 1 to index 21 shown in Table 4, because the other five message types are not used by default. Thus, Fig. 3 includes 21 message types which are enabled by the default Bitcoin implementation.

Analyzer To test incoming networking traffic, LION stores the message traffic in testing datasets and tracks the frequency occurrences of each message types, Λ_{test}. LION then compares the normalized frequency distribution from testing dataset, Λ_{test}, with the reference profile from the training dataset, Λ_{train}, by computing the similarity in the correlation coefficient, ρ. Fig. 3 presents these ρ values when the Analyzer compares the tested traffic with the training-generated reference distribution. The *Normal* case shows the highest ρ at 0.993 (Fig. 3(a)). The *Abn-1p* also shows high similarity with the reference distribution at ρ = 0.946 (Fig. 3(b)) because *Abn-1p* decreases the number of peer connections, and the frequencies are normalized, indicating that the inter-frequency between the Bitcoin message types a peer sends is similar from one peer to another. By contrast, *Abn-Syn* significantly shifts the distribution and lowers the correlation at $\rho = 0.074$ as it increases the `Block` message (with index 9 in Fig. 3(c)). *Abn-DoS* floods the `PING` message (with index 12 in Fig. 3(d)) to shift the frequency distribution, resulting in ρ = 0.02.

Our insights for each of the networking anomalies extend to multiple testing dataset samples. Fig. 4 shows the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing



(a) TN and FP for *Normal* (b) TP, FN for *Abn-1p*, *Abn-Syn*, *Abn-DoS* (c) Accuracy, Precision, Recall and F1-score

Fig. 6. Detection performances for controlling τ_n using the x factor (x increases by 0.01 each time).

cumulative distribution function (CDF) of the ρ between Λ_{train} and Λ_{test} for the small ρ values for *Abn-DoS* and *Abn-Syn* (Fig. 4(a)) and for the large ρ values for *Normal* and *Abn-1p* (Fig. 4(b)). These measurements across multiple testing samples inform the selection for τ_Λ to be compared with ρ between Λ_{train} and Λ_{test}; any threshold $\tau_\Lambda \in [0.9605, 0.9857]$ can be used to correctly distinguish *Normal* and *Abn-1p* (and by extending *Abn-Syn* and *Abn-DoS*, which shift the Λ_{test} even greater from Λ_{train} than *Abn-1p* and thus have even smaller $\rho < 0.125$).

5.4 Analysis of Timing Windows, T_{train} and T_{test}

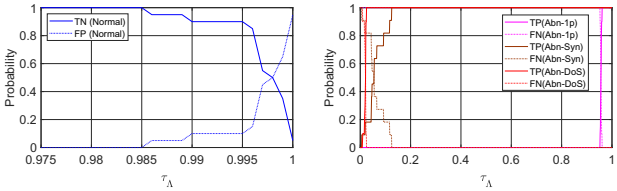
The timing windows for training and testing, T_{train} and T_{test} respectively, affect the LION performances. More specifically, increasing the timing window sizes increases the detection performances although there is a tradeoff in cost-efficiency (i.e., greater dataset sizes requires more processing as we will later study in Section 7.1). Fig. 5 increases the T_{test} and T_{train} and plots the ρ between the training-generated reference and the testing of *Normal*. Because it tests the normal case, the greater the ρ is, the better the detection performance gets. The ρ increases with both the T_{test} and T_{train}. At the T_{test} and T_{train} being the greatest in our experiment at 1000 minutes and 2000 minutes, respectively, we observe the maximum $\rho > 99.9\%$. However, $\rho \neq 100\%$ even if larger T_{test} and T_{train} are used because of the natural randomness in the Bitcoin networking, and the training and the testing are done in independent and non-overlapping times. Because we see that when T_{test} is fixed at 100 minutes, then train window from 80 minutes to 2000 minutes remain almost constant, so we use T_{test}=100 minutes for the LION performance evaluation.

6 LION Detection Performances

In this section, we evaluate the LION's detection performance by testing multiple datasets, and also show the experimental results about detection performance.

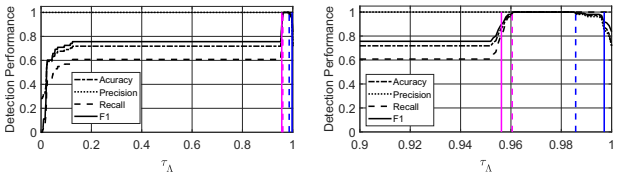
6.1 Detection Performance: Networking Threats

We evaluate the detection performance in terms of the confusion matrix, i.e. true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Based on them, we derive the metrics of Accuracy, Precision, Recall and F1-score (F1 for short in this paper) while varying the detection thresholds. We focus on presenting the detection performance for τ_Λ and τ_n individually and omit the evaluation of τ_s, since analyzing τ_s is similar to the results of τ_n and provides limited values for the anomaly detection against our testing scenarios (see Section 6.1.3). We then combine the



(a) *Normal* (b) *Abn-1p*, *Abn-Syn*, *Abn-DoS*

Fig. 7. TP, TN, FP, FN while varying τ_Λ.



(a) Original plot (b) Zoom-in plot

Fig. 8. Detection performance evaluations while varying τ_Λ. The four vertical lines from left to right correspond to the the average and maximum ρ of *Abn-1p*, the minimum and average ρ of *Normal*. The middle two dotted lines correspond to yield zero-error.

parameters for the final detection decision, which provides high detection performance in our testing.

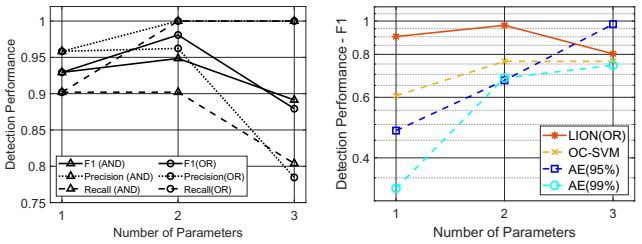
6.1.1 Evaluation for τ_n

Fig. 6 shows the detection performance for varying τ_n. We introduce x as a range ratio for tuning τ_n to generate the range τ_n' for LION, which is defined as: $\tau_n' = [\tau_{n,min} + \frac{1-x}{2} \cdot (\tau_{n,max} - \tau_{n,min}), \tau_{n,max} - \frac{1-x}{2} \cdot (\tau_{n,max} - \tau_{n,min})]$. For example, if $x = 0.5$, τ_n' is half the length as the τ_n range while the center of τ_n is fixed.

Fig. 6(a) presents the performance testing the *Normal* case and because it is given Normal datasets, we either have TN or FP. TN increases and FP decreases as x increases because more testing will be decided as normal when increasing x. By contrast, Fig. 6(b) shows the anomaly cases, which yields TP and FN. TP decreases and FN increases as x increases because there are greater occurrence for deciding that the tested traffic is normal with increasing x.

Fig. 6(c) combines the testing cases and measures the Accuracy, Precision, Recall, and F1-score with increasing x. We observe that the trade-off between Precision (focusing on FP) and Recall (focusing on FN) as Precision increases with x increasing while Recall decreases with x increasing. In other words, as we become more conservative in detecting anomalies with increasing x (less probability to detect anomalies), FP decreases (resulting in Precision increasing) and FN increases (resulting in Recall decreasing). We prioritize the F1-score as our evaluation metric because it is the harmonic mean between Recall and Precision. We choose $x = 0.7$ yielding the maximum F1-score (see Fig. 6(c)), i.e.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing



(a) Using multiple parameters (b) F1-score

Fig. 9. Detection performance: parameter analysis (Figure 9(a)) and the comparison with semi-supervised ML.

when using the n for LION, we use 70% ($x = 0.7$) of τ_n for the rest of the paper.

6.1.2 Evaluation for τ_Λ

Fig. 7 presents the detection performance while varying τ_Λ for various testing datasets. Fig. 7(a) shows the performance of *Normal*, in which TN decreases and FP increases with increasing τ_Λ. Fig. 7(b) shows the abnormal cases, where TP increases and FN decreases with increasing τ_Λ.

Fig. 8 shows the Accuracy and F1-score performances while varying τ_Λ. Unlike the case with τ_n, we can select τ_Λ so that it maximizes all of Accuracy, Precision, Recall, and F1-Score. Regarding τ_n, there are some overlaps between the normal and abnormal cases, which causes false detection, i.e., leaning in FP or FN depends on the user, which results in the mutual constraint between Precision and Recall. Whereas, τ_Λ can drastically distinguish the normal and abnormal cases, and there is no overlap between them, so it does not suffer the mutual constraint. The Λ comparison and the corresponding ρ provides a strong indicator for LION, and there is no tradeoff between Precision and Recall, i.e., we can choose τ_Λ to maximize the performances according to all four metrics. According to our experiments, *Abn-1p* provides the hardest case for detection (because it merely reduces the connection diversity where the Eclipse attacker behaves normally). In Fig. 8(b), we can select any threshold value τ_Λ (for comparing it with the ρ) between the two vertical dash lines, and such selection will provide optimal performances; we therefore choose such $\tau_\Lambda \in [0.9605, 0.9857]$ and more specifically $\tau_\Lambda = 0.9731$, which is the middle center in that range.

6.1.3 Evaluation for Combining the Parameters

We use multiple measurable parameters from the networking traffic, i.e., n, s and Λ. In this section, we combine their use for the eventual detection decision. We provide two mechanisms for combining them: the logical AND across the three parameters ("AND") and the logical OR across the three parameters ("OR"). For example, if one or more of the parameters decides that there is anomalous, then the "OR"-based LION decides that it detects an anomaly.

As seen in Fig. 2(c), τ_n is not able to fully classify *Abn-Syn* and *Normal*, but τ_Λ can easily distinguish them. However, it increases the FP rate when testing the *Normal* because, as long as one parameter considering it as abnormal, LION detects that networking traffic as abnormal.

Fig. 9(a) presents the detection performances while using different combinations of our parameters for LION. The horizontal axis indicates the number of parameters used where #"1" means only applying n (and τ_n) for detection, #"2" denotes combining n (and τ_n) and Λ (and τ_Λ) for detection, and #"3" stands for combining all of them, i.e., n (and τ_n), Λ (and τ_Λ) and s (and τ_s) for executing the LION Analyzer. Because Precision focuses on the FP inverse impact and Recall focuses on the FN inverse impact, Fig. 9(a) shows that, when using the logical OR combination on Recall, the FN rate reduces which increases the Recall. Conversely, when using the logical AND combination, the FP rate decreases and the Precision increases. Using the two parameters of n and Λ and combining the decisions through logical OR provides us with the highest F1-score performance at 97.18% in our testing cases, including both normal and abnormal cases. Because adding additional parameters do not necessarily improve the detection performance but will increase the analysis complexity and make LION heavyweight, we stop from further increasing the parameter complexity and focus on the two parameters, n and Λ.

In addition, we compare the detection performance of LION with semi-supervised ML in term of using multiple parameters. The ML methods use the same datasets as input to train and test the model in terms of the same parameter combination, i.e., #"1" means only using the data reflecting n, #"2" means using the data reflecting both n and Λ, and #"3" means using the data reflecting all of n, Λ and s. Regarding AE, we consider two AE models with different threshold ranges of 95% and 99% to detect outliers. AE(95%) rejects 5% extremes while AE(99%) rejects 1% only. 95% and 99% are widely being used in statistics for measuring confidence intervals [23]. More details about the implementation of the ML methods refer to Section 7.2.

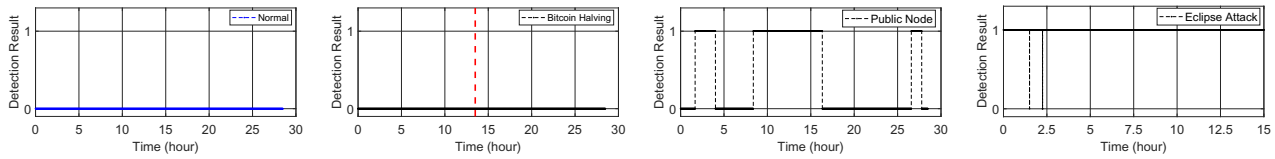
Fig. 9(b) present F1-score measurements respectively in terms of using different number of parameters to reach the optimal performance. LION provides 97.18% F1-score using two parameters, OC-SVM provides the same 76.34% F1-score using both two and three parameters, AE(95%) provides 67.22% F1-score using two parameters and 97.92% F1-score using three parameters, and AE(99%) provides 68.3% F1-score using two parameters and 74.37% F1-score using three parameters. Thus, LION provides 97.18% F1-score which is comparable to the best-case ML providing 97.9%.

6.2 Detection Performance: Other Anomaly Events

We build LION on the previous analyses and control, e.g., $\tau_\Lambda = 0.9731$, $\tau_n = [27239, 36917]$ (the same optimal choice but proportionally greater by T_{test}, and $x = 0.7$ is applied) and $T_{test} = 100$ minutes (a sliding time-window for testing is used with the overlapping time windows), and run it against the scenarios described in Section 4.4. The whole testing/observing time is 30 hours for *Normal*, *Bitcoin Halving*, *Public Node* respectively, and 15 hours for *Eclipse Attack*. The timing window $T_{test} = 100$ minutes slides forward during the testing time (15 or 30 hours). The detection results are shown in Fig. 10. The *Normal* is collected the same way as before, and LION decides normal and thus no anomaly detected, i.e., the detection probability is 0, as shown in Fig. 10(a).

Fig. 10(b) shows the detection result of *Bitcoin Halving*. LION was monitoring the traffic while the Bitcoin Halving event occurred (which time is indicated by the red dotted vertical line in the plot). LION decided normal networking

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing



(a) Normal with Pr(Detect)=0 (b) Bitcoin Halving with Pr(Detect)=0 (c) Public Node with Pr(Detect)=0.4031 (d) Eclipse Attack with Pr(Detect)=0.9998

Fig. 10. LION detection results: 0 indicates normal and 1 indicates anomaly; Pr(Detect) denotes the detection probability.

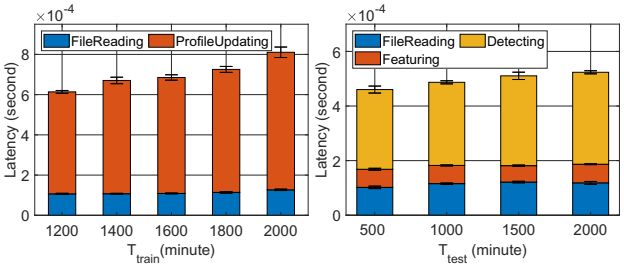
(no anomaly detection) despite the significant shift in the block reward, indicating that there are no noticeable changes in the Bitcoin networking (generally affected by the transaction generation and Bitcoin commerce) immediately before or after the Bitcoin Halving event on May 11th, 2020. To put the time scale of our testing in to perspective, the Bitcoin protocol changes its mining block difficulty (determining the load of the miner network in the expected number of hash computations to find a valid block) every two weeks, which is longer than our testing period in the scale of hours. Bitcoin halving is a real-world event, and we also had no idea if it would bring any message traffic anomaly before we used LION to examine it. That means that we can use LION to examine the event that we do not know its ground-truth, which shows a practical use of LION. If such an event is even not detected as anomaly by an anomaly detector (which often has certain false positive ratio), i.e., by our LION, we can say this event is normal in the perspective of the application-layer message traffic.

Fig. 10(c) presents the detection result when we use LION with a public node changing the node setup from the training. The public node not only has outbound connections but also accepts inbound connection requests from Mainnet. The inbound connections are dynamically changing, so n and Λ of the collected data fluctuate dynamically, thus triggering LION to detect anomaly when the peer connections diverge too much from the reference training. For instance, we observed a correlated behavior between the LION detecting anomalies and the frequency of the "invalid block" messages to the LION node; more "invalid block" messages indicating the misbehaving peers getting banned were observed when LION detected anomaly ("1") than normal ("0"). The detection probability was 40.31%, i.e., LION detected anomaly 40.31% of the testing instances.

Fig. 10(d) shows the LION detection when the LION node is under Eclipse attack where the attacker is randomly dropping half the packets. LION detected anomaly with a detection probability of 99.98% since the Eclipse attack changed the networking.

7 LION Cost-Efficiency Performances and Miner System Analysis

In this section, we analyze the computational cost overhead of LION in processing, e.g., CPU usage, memory usage, and time latency. We further include the system analysis to measure how the LION detection implementation affects the other Bitcoin node performance. More specifically, we study how LION minimally affects the mining operation when it is enabled and run in parallel. The analysis is important because mining yields the reward profit and incentives the participation to the blockchain operations. To further highlight the efficiency of LION, we compare its efficiency with the ML-based detection, adopted by previous research (see Section 9).



(a) Training: Profile Updating (b) Testing: Anomaly Detecting

Fig. 11. LION latency with ascending timing windows.

7.1 Cost-Efficiency Performance

We test LION's computational overhead to demonstrate its cost efficiency and to establish the practicality of the LION deployment to the cryptocurrency miners.

We present the cost efficiency results with respect to the timing windows. We separate the tasks within training and testing and generate 10,000 experimental samples by running them repeatedly and include the 95% confidence interval in the plots. One of the dominant factors for the cost overhead is the dataset/file reading after the Monitor stores them. For the training, in Fig. 11(a), The File Reading grows relatively slower than The Profile Updating, the latter of which is after the File Reading and includes deriving τ_Λ and τ_n. For the testing, in Fig. 11(b), the growth of the tasks is even less sensitive to the timing windows. The File Reading of the testing datasets is the dominant cost factor compared to the Featuring and the Detection. The Featuring (for deriving the τ_Λ and τ_n) costs less processing time than the Detection (which computes, e.g., ρ, compares, and combines the multiple parameters). The aggregate cost for testing is $5.23 \cdot 10^{-4}$ seconds. The CPU processing usage (per processor) is 0.6% and the MEM usage is 30.72 MB, which includes testing and training given our machine specification in Table 3.

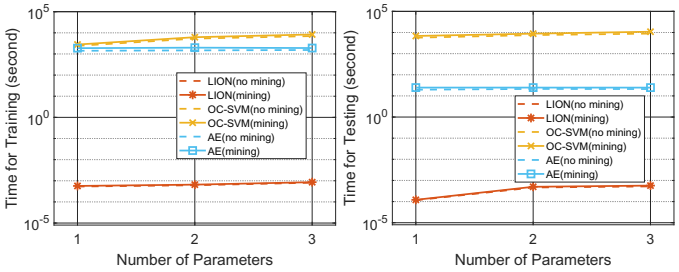
7.2 System Analysis for Mining Impact and Comparison with Machine Learning

We analyze the mining impact by the LION implementation and compare LION with the machine learning (ML)-based approaches for detection. Section 9 discusses the related work using ML for securing cryptocurrency, motivating our trial implementations based on ML. As discussed in Section 2, we focus on the comparison with semi-supervised ML, in particular, with One-class SVM (OC-SVM) [24]–[26] and AutoEncoder (AE) [23], [24], because the popular anomaly detection approaches in particular use OC-SVM [11]–[13] and AE [14] in the context of cryptocurrency blockchains.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing

TABLE 5
Resource usage: LION vs Semi-supervised ML.

	LION	OC-SVM	AE
CPU (%)	0.6 (1x)	12.5 (20.83x)	23.75 (39.58x)
MEM (MB)	30.72 (1x)	1689.6 (55x)	1740.8 (56.67x)



(a) Time for training (b) Time for testing

Fig. 12. The comparison of time consumption. The solid line shows the time overhead while running the mining in parallel, and the dotted line corresponds to the time overhead while the mining is disabled.

The comparison between LION and ML methods is fair in the machine setup and the datasets including the data that reflect the three parameters (i.e., n, s and Λ as for LION) used in the algorithms. In other words, the modular nature of LION enables us to swap it with the other ML algorithms and we tested the performance comparison on the same platform building on our Bitcoin node implementation.

We configured and fine-tuned the ML algorithms using the Bitcoin networking data for its analyses. To provide more details, for OC-SVM, we chose the radial basis function (RBF) kernel to consider non-linearity with the reciprocal of the number of features for the kernel coefficient. For AutoEncoder, we accept the instance in question as normal if its reconstruction error is located within the pre-calculated distribution from the normal data, while any instance out of the distribution is regarded as an anomaly. To consider outliers in the normal profile, we reject both extremes (either 1% or 5% in two experimental settings, respectively), rather than taking the minimum and maximum values in the distribution as the threshold. The implementation details of the AutoEncoder model is as follows. The AutoEncoder model has been configured with three hidden layers (since there are three parameters, which require three hidden layers) and the number of neurons at each layer was chosen based on the number of features: $\langle \frac{1}{2}, \frac{1}{4}, \frac{1}{2} \rangle \times$ number of features, from the first hidden layer to the last layer. We used a mixture of `ReLU` and `tanh` as the activation of AE for non-linearity with Mean Squared Error (MSE) to calculate the loss.

Table 5 about the resource usage for training shows the CPU consumption of the detection engines up to the significant digits. LION's processing consumption in CPU is 0.6% (per CPU), which is substantially smaller than the ML algorithms whose CPU consumptions are between 20 times and 40 times greater than LION. Even with the more CPU-efficient OC-SVM, it consumes 12.5% (per CPU) which could have otherwise been used for mining. Also, the ML algorithms consume more memory by 55-57 times than LION; such memory efficiency of LION is more helpful to the newer-generation cryptocurrencies, e.g. Ethereum [27] incorporates proof-of-memory to the PoW to discourage the use of ASIC hardwares for the decentralization of mining.

Fig. 12 compares the time consumption for processing the detection while simultaneously mining ("mining") and not ("no mining"). More specifically, the time consumption is measured from the time that LION accepts the inputs to the time that LION creates the reference profile for model training or to the time that LION makes detection decision for model testing, and both of them are based on the default implementation without optimizing the CPU processing (e.g., if we optimize the processing, then we may be able to reduce the time consumption). Fig. 12(a) compares the LION's and the semi-supervised approaches' time latency for training the detection model. The LION's training-time cost is seven orders of magnitude smaller than the semi-supervised approaches. Fig. 12(b) shows that the LION's processing-time cost for testing is five orders of magnitude smaller than the more time-efficient algorithms, AE, while eight orders of magnitude smaller than the less time-efficient one, OC-SVM. LION takes ten-thousandth of seconds (i.e., on the scale of 1/10,000 seconds), the AE executions take tens of seconds and OC-SVM takes tens of minutes. It is worth recalling that the measured time is the aggregated result for training and testing the "entire" entries within the timing window, which means that this is not a "per-entry" cost. Thus, the ML methods need hours to process the entire entries. Furthermore, we notice that between the two ML methods, OC-SVM spends more time to process the training and testing data (which corroborates the result presented in [28]), even though it uses less computation resource than AE in terms of Table 5. As state in [29], SVM is known to be an time expensive algorithm.

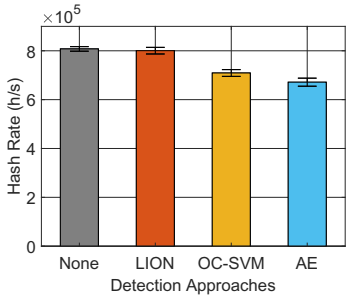


Fig. 13. The comparison of mining rate reduction.

Fig. 13 compares the Bitcoin mining rate for our Bitcoin node implementation when there is no detection mechanism running in parallel (the miner at "None" computes $8.077{\cdot}10^5$ h/s), LION running in the background (the miner computes $8.00{\cdot}10^5$ h/s), OC-SVM running ($7.091{\cdot}10^5$ h/s), and AE running ($6.714{\cdot}10^5$ h/s). Because PoW is computational-power-fair, i.e., the chance of finding the valid block and earning the reward is proportional to the mining rate in h/s, the financial mining reward reduction is less than 0.95% for LION while that is 12.21% for OC-SVM and 16.88% for AE. Compared to LION having less than 1% financial reward reduction, the reward reduction cost is vitally greater than OC-SVM and AE. Fig. 13 is consistent with Table 5 showing OC-SVM uses less computational resources so as to reduce less mining rate.

LION is drastically superior to the existing ML-based detection mechanisms in the execution costs in computations, memory, and time latency. Efficiency and low costs in such resources is critical for the practicality of LION to deploy it in the miner nodes, since such resources are valuable for miners due to their participation in the resource-intensive distributed consensus protocol. For miners with high com-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2022.3152937, IEEE Transactions on Dependable and Secure Computing

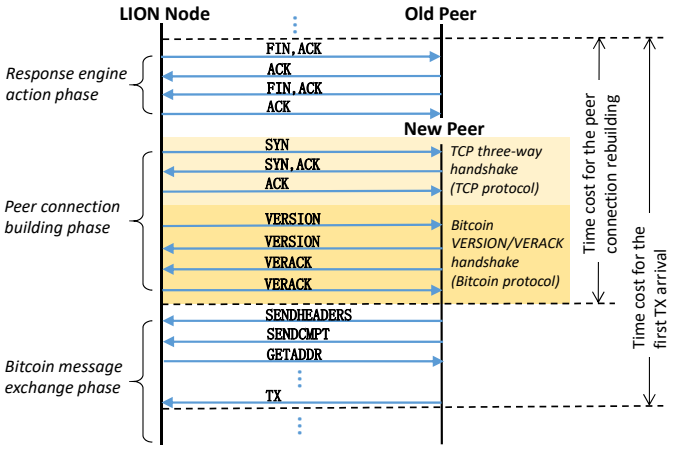


Fig. 14. Illustration of peer connection rebuilding: the connection failure may happen in either the TCP three-way handshake or the Bitcoin `VERSION/VERACK` handshake. If the connection building fails, the LION node will look up another IP address from its peer table to have another try of peer connection. If there is no failure during these two handshakes, the peer connection buidling phase succeeds, and the Bitcoin message exchange phase will follow up.

putational power resources (such as those based on GPU and ASIC), ML-based anomaly detection may become feasible, as increasing the system computing power can reduce the relative detection cost; however, LION provides a much more lightweight detection scheme which can extend its application/use more generally to other miners, contributing to further decentralization of the mining ecosystem [30], [31]. Making LION deployment-friendly for such existing miners (e.g., our prototype builds on the miner implementation based on Bitcoin Core) enables us to leverage the existing Bitcoin ecosystem and to forgo introducing separate entities implementing security.

8 Building Towards Active Defense

The anomaly detection is a passive operation, rather than the active operations for mitigation or prevention. This section presents and discusses the potential active responses based on the peer-connection control to protect the Bitcoin node extending the anomaly detection.

Our response engine uses the reactive response approach for mitigating and recovering from the networking threats on the victim node. The reactive response approach means that the Bitcoin node takes the action after LION detects the anomaly as a response to the detection. Such reactive response mechanism builds on detection and provides greater cost-effectiveness than more proactive approaches such as dynamic randomization based on moving-target defense. In our active defense, once detecting anomaly, the miner drops all the connections, and reconnects to freshly and randomly selected peers. This eliminates the potential vulnerability for exploiting the individual connections, in contrast to the alternative approaches of prioritizing and distinguishing between the connections.

We implement the response engine by integrating it with LION and the Bitcoin node operations. The active measures from the response engine disconnects and then re-establishes the Bitcoin connections with a new set of random peers via the Bitcoin console. The LION node controls the

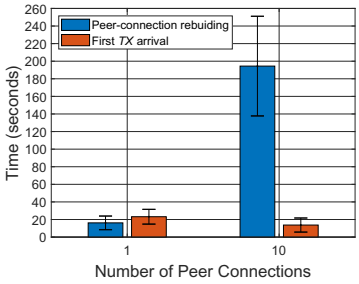


Fig. 15. Time cost for peer reestablishment as active countermeasure.

outbound connections for the active defense, and there are ten connections by the default Bitcoin setting.

Fig. 14 illustrates the peer connection rebuilding after the response engine takes action. Once the old peer (detected as anomaly) gets dropped, the LION node starts to reach out to new peer. The peer connection building phase includes the TCP three-way handshake and the Bitcoin `VERSION/VERACK` handshake, but both could fail due to any availability or networking issues. Thus, the peer connection rebuilding will not succeed until the LION node eventually find a new peer that can finish the two handshakes with it. With that, we define the time cost for the peer connection rebuilding as from the time point that the response takes action to the time point that the eventual new peer finishes the two handshakes. Also, we define the time cost for the first `TX` arrival as from the time point that the response takes action to the time point that the LION node receives the first `TX` sent by the eventual new peer.

Fig. 15 shows the time cost overhead for the peer re-establishment from the time that LION triggers the response engine to the time that the connection(s) are re-established. We test when the Bitcoin node has 10 outbound peer connections which is set as default and when the Bitcoin node has only 1 outbound peer connection for comparison. The experimental results show that the Bitcoin node takes 16.18 seconds on average for reestablishing one new outbound peer connection, while it uses 194.43 seconds on average for rebuilding 10 outbound peer connections after the reactive response is triggered via LION. Such re-establishment requires trial-and-error in selecting/establishing the new peer connections, as not all peers are available at the time of the connection request, and thus typically requires transmitting multiple initial requests. As seen in Fig. 14, once the other peer responds, the new connection gets established via the TCP three-way handshake and via the Bitcoin `VERSION/VERACK` handshake. When the connection gets re-established, the LION node can begin receiving Bitcoin-relevant and beneficial communications. Having greater connections make the receiving time quicker as there is greater connectivity to the rest of the Bitcoin Mainnet. In our experiment, it takes 23.17 seconds for the Bitcoin node to receive the first viable `TX` message after the response starts when there is only one peer connection, while it takes 13.71 seconds to get the first effective `TX` message after the response begins when there are 10 peer connections.

Though the peer reconnection as the active countermeasure is not free, it is worth doing that. Because it is temporary loss vs. permanent loss: if we don't drop the malicious peer connections to the miner node, the miner would even continue to waste mining power (e.g., mining on the out-of-date block due to block withholding), and

leads to extremely more loss of mining reward. In practice, there is no really free security protection approach, and every security approach has its own cost. However, as long as the whole system can save more cost by using the security approach than the system without using the security approach (suffering more cost owing to the attack), then the security approach should be adopted. In addition, the reconnection process actually enforces the LION-using node to establish healthy peer connections to the rest of the P2P network timely, which will make the P2P network converge into a stable state.

9 Related Work

We perform a related work review including the anomaly detection in cryptocurrency, the statistical analysis for deanonymizing Bitcoin users, and the Bitcoin message analysis for discovering P2P topology.

Anomaly detection in cryptocurrency Successfully applied to anomaly detection in general computer networking, e.g., [23], [32], [33], machine learning (ML) has been proposed to secure cryptocurrency. Previous literature suggests using machine learning for anomaly detection in Bitcoin and other cryptocurrencies, including a comprehensive survey in Rahouti et al. [34].

In supervised environment knowing what to detect and being able to train the decision engine accordingly, previous research included the investigation of the anomalies created by cybercrime activities [35] and using Bitcoin transaction information to infer the user behavior [10]. using ML to de-anonymize the entities in Bitcoin [9]. However, more closely related to our model is the unsupervised or semi-supervised ML algorithms where the data for the detection training are limited; such approaches also enable the detection of unknown anomalies or zero-day attacks. In such an environment, the research projects [11], [12] applied unsupervised ML to detect suspicious users and transactions in Bitcoin transition network. They typically used One-Class Supported Vector Machine (OC-SVM) to extract features from the unlabeled data for anomaly detection and used k-means to help cluster similar features into groups (note that k-means itself is not a method for anomaly detection). Similarly, in [13], OC-SVM was used to detect the anomalies in Bitcoin transactions collaborating with k-means clustering to group similar outliers. SquirRL [36] used a deep reinforcement learning to detect selfish mining and block withholding attacks in Bitcoin, which introduced an agent to implement the security mechanism and monitor the errors and the block rewards. Also, Scicchitano et al. [14] proposed an Ensemble Deep Learning to detect deviant behaviors on Ethereum [27] blockchain, where the base learner is the AutoEncoder (AE). Most recently, Kim et al. [37], [38] also proposed an anomaly detection engine on top of AE, which is able to effectively detect the message-based DoS attack. However, such a ML-based detection engine is computationally intensive, which often suffers the long-term training and testing latencies and effects the mining operation.

These previous research motivate us to explore using learning approach for anomaly detection against the cryptocurrency networking threats. However, due to the challenges of such ML-based detection for cryptocurrency applications, we propose LION, an efficient statistical-analyses-based anomaly detection engine.

Statistical analysis for deanonymizing Bitcoin users We reviewed the prior research using statistical analysis on Bitcoin transaction traffic. Ron and Shamir [39] analyzed the statistical properties of the Bitcoin transactional graph. In [40], the authors collected and analyzed the real-time transaction traffic to create the mapping between Bitcoin addresses to IP addresses, i.e., to deanonymize Bitcoin users. Neudecker and Hartenstein [41] analyzed the P2P network traffic to see whether it can also be useful in the deanonymization of Bitcoin users. According to their discovery, a small number of participants exhibit correlations. It makes them susceptible to network-based deanonymization attacks. In [42], the authors focused on the propagation timing information and clustering-based mechanism using such information for the Bitcoin user deanonymization. These literature used the transaction message traffic to deanonymize Bitcoin users, in contrast to LION in its purpose of anomaly detection using different messages.

Bitcoin message analysis for discovering P2P topology Other research utilizes Bitcoin networking messages to infer the P2P topology, which includes using the transaction accumulation of double-spending transactions [43] and the orphan transactions which arrive out of order [44]. Grundmann et al. [43] aimed at inferring the topology of Bitcoin P2P network by exploiting transaction accumulation of double-spending transactions. Delgado-Segura et al. [44] further proposed a novel technique called TxProbe, providing a stronger result with full network topology using orphan transactions (that arrive out of order). Comparing with that, Grundmann et al. only targeted neighborhood discovery. Arthur et al. [45] presented the hourly traffic distribution of a Bitcoin node concerning three message types (`INV`, `TX` and `BLOCK`) to demonstrate their observation that the transmission of blocks consumes most of the bandwidth. However, their research mainly focus on discoveries that an adversary can exploit the current scalability measures to delay the transactions/blocks propagation to specific nodes instead of doing analyses over various message types.

10 Conclusion

Bitcoin and other cryptocurrencies are permissionless and trustless by design, and their underlying P2P network lacks the cryptographic protection and the trust in the peer's identifiers. Cryptocurrencies rely on the P2P network to provide the information to drive the rest of the cryptocurrency operations. Any message traffic anomalies should be concerning for bitcoin/cryptocurrency node. Not only the network-layer attacks incurred by the unprotected P2P network can be reflected by the anomalous message traffic, but also the message traffic state change involving anomalous block/transaction messages arrival will definitely impact the Bitcoin node mining efficiency no matter what event is causing it. Due to such a vital importance of the message traffic on the P2P network of cryptocurrencies, we propose LION to build anomaly detection using networking traffic information and statistical analysis so that LION is efficient for the permissionless blockchains. LION focuses on anomaly detection on the application-layer message traffic, while we broaden its use to examine the message traffic

state when some other typical events occur in order to show whether the event brings anomaly or not in the message traffic perspective.

We implement LION on the active Bitcoin miner node connected to the the Bitcoin Mainnet and test it with the data collected from both the Mainnet and from our anomaly prototypes. The experimental results show that LION is efficient against networking anomalies and gains high detection performance, i.e., 97.18% F1-score. Furthermore, LION has a significantly superior cost-efficiency in computation, execution time, and mining reduction cost (when LION runs alongside mining), compared to the machine learning (ML)-based approaches (adopted in the state-of-the-art for securing blockchain networking). Our experimental results show that the mining rate reduction (and therefore the financial block reward reduction) is limited to 0.95% for LION, while the reduction costs for running the ML-based detection is 12% or greater. Such cost efficiency is critical to enable its deployment on the existing miners (whose computational resources are of premier importance), as opposed to introducing separate entities for implementing the anomaly detection.

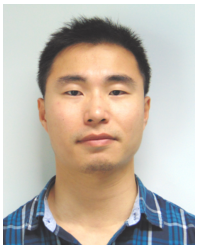
Acknowledgments

This research was supported in part by Colorado State Bill 18-086 and by the National Science Foundation under Grant No. 1922410.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[2] CoinMarketCap, "Cryptocurrency prices, charts and market capitalizations," 2021. [Online]. Available: https://coinmarketcap.com/
[3] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.
[4] M. Tran, I. Choi, G. J. Moon, A. V. Vu, and M. S. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *IEEE Symposium on Security and Privacy (IEEE S&P)*, 2020.
[5] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2016.
[6] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (S&P)*, 2017.
[7] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*, 2013.
[8] M. Vasek, M. Thornton, and T. Moore, "Empirical analysis of denial-of-service attacks in the bitcoin ecosystem," in *Financial Cryptography and Data Security*, 2014.
[9] M. Harlev, H. Sun Yin, K. Langenheldt, R. Mukkamala, and R. Vatrapu, "Breaking bad: De-anonymising entity types on the bitcoin blockchain using supervised machine learning," in *51st Hawaii International Conference on System Sciences*, 2018.
[10] H. Tang, Y. Jiao, B. Huang, C. Lin, S. Goyal, and B. Wang, "Learning to classify blockchain peers according to their behavior sequences," *IEEE Access*, vol. 6, 2018.
[11] J. Hirshman, Y. Huang, and S. Macke, "Unsupervised approaches to detecting anomalous behavior in the bitcoin transaction network," *3rd ed. Technical report, Stanford University*, 2013.
[12] T. Pham and S. Lee, "Anomaly detection in bitcoin network using unsupervised learning methods," *arXiv preprint arXiv:1611.03941*, 2016.
[13] S. SAYADI, S. B. REJEB, and Z. CHOUKAIR, "Anomaly detection model over blockchain electronic transactions," in *15th International Wireless Communications Mobile Computing Conference (IWCMC)*, 2019.
[14] F. Scicchitano, A. Liguori, M. Guarascio, E. Ritacco, and G. Manco, "Deep autoencoder ensembles for anomaly detection on blockchain," in *25th International Symposium on Methodologies for Intelligent Systems*, 2020.
[15] P. Chatzigiannis and K. Chalkias, "Proof of assets in the diem blockchain," in *Applied Cryptography and Network Security Workshop on AIBlock*, 2021.
[16] W. Fan, S.-Y. Chang, S. Emery, and X. Zhou, "Blockchain-based distributed banking for permissioned and accountable financial transaction processing," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020.
[17] BitcoinProject, "The set of 26 bitcoin message types," 2020. [Online]. Available: https://developer.bitcoin.org/reference/p2p_networking.html
[18] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," *Commun. ACM*, vol. 61, no. 7, 2018.
[19] M. Rosenfeld, "Analysis of bitcoin pooled mining reward systems," *CoRR*, vol. abs/1112.4980, 2011. [Online]. Available: http://arxiv.org/abs/1112.4980
[20] Y. Kwon, D. Kim, Y. Son, E. Vasserman, and Y. Kim, "Be selfish and avoid dilemmas: Fork after withholding (faw) attacks on bitcoin," in *2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.
[21] S.-Y. Chang, Y. Park, S. Wuthier, and C.-W. Chen, "Uncle-block attack: Blockchain mining threat beyond block withholding for rational and uncooperative miners," in *Applied Cryptography and Network Security*, 2019.
[22] W. Fan, S. Chang, X. Zhou, and S. Xu, "Conman: A connection manipulation-based attack against bitcoin networking," in *IEEE Conference on Communications and Network Security (CNS)*, 2021.
[23] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "Gee: A gradient-based explainable variational autoencoder for network anomaly detection," in *2019 IEEE Conference on Communications and Network Security (CNS)*, 2019.
[24] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, 2007.
[25] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *ACM SIGKDD Workshop on Outlier Detection and Description*, 2013.
[26] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018.
[27] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, 2014.
[28] W. Fan, J. Kim, I. Kim, X. Zhou, and S.-Y. Chang, "Performance analyses for applying machine learning on bitcoin miners," in *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, 2021.
[29] F. Nie, W. Zhu, and X. Li, "Decision tree svm: an extension of linear svm for non-linear classification," *Neurocomputing*, vol. 401, 2020.
[30] V. Buterin, "A next-generation smart contract and decentralized application platform," *Ethereum project white paper*, vol. 3, no. 37, 2014.
[31] L. Luu, Y. Velner, J. Teutsch, and P. Saxena, "Smartpool: Practical decentralized pooled mining," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017.
[32] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *2010 IEEE Symposium on Security and Privacy (S&P)*, 2010.
[33] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.
[34] M. Rahouti, K. Xiong, and N. Ghani, "Bitcoin concepts, threats, and machine-learning security solutions," *IEEE Access*, vol. 6, 2018.
[35] H. Sun Yin and R. Vatrapu, "A first estimation of the proportion of cybercriminal entities in the bitcoin ecosystem using supervised machine learning," in *2017 IEEE International Conference on Big Data (Big Data)*, 2017.
[36] C. Hou, M. Zhou, Y. Ji, P. Daian, F. Tramer, G. Fanti, and A. Juels, "Squirrl: Automating attack discovery on blockchain incentive mechanisms with deep reinforcement learning," *arXiv preprint arXiv:1912.01798*, 2019.
[37] J. Kim, M. Nakashima, W. Fan, S. Wuthier, X. Zhou, I. Kim, and S.-Y. Chang, "Anomaly detection based on traffic monitoring for secure blockchain networking," in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney Australia, May 2021.

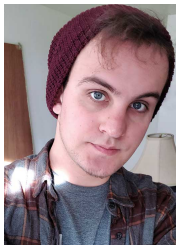
[38] J. Kim, M. Nakashima, W. Fan, S. Wuthier, X. Zhou, I. Kim, S.-Y. Chang, "A machine learning approach to anomaly detection based on traffic monitoring for secure blockchain net-working," *IEEE Transactions on Network and Service Management*, 2022. Accepted.
[39] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *International Conference on Financial Cryptography and Data Security (FC)*, 2013.
[40] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *International Conference on Financial Cryptography and Data Security (FC)*, 2014.
[41] T. Neudecker and H. Hartenstein, "Could network information facilitate address clustering in bitcoin?" in *International Conference on Financial Cryptography and Data Security (FC)*, 2017.
[42] A. Biryukov and S. Tikhomirov, "Transaction clustering using network traffic analysis for bitcoin and derived blockchains," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2019.
[43] M. Grundmann, T. Neudecker, and H. Hartenstein, "Exploiting transaction accumulation and double spends for topology inference in bitcoin," in *International Conference on Financial Cryptography and Data Security (FC)*, 2019.
[44] S. Delgado-Segura, S. Bakshi, C. Pérez-Solà, J. Litton, A. Pachulski, A. Miller, and B. Bhattacharjee, "Txprobe: Discovering bitcoin's network topology using orphan transactions," in *International Conference on Financial Cryptography and Data Security (FC)*, 2019.
[45] A. Gervais, H. Ritzdorf, G. O. Karame, and S. Capkun, "Tampering with the delivery of blocks and transactions in bitcoin," in *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.



Wenjun Fan is currently an assistant professor with Xi'an Jiaotong-Liverpool University (XJTLU). Before joining XJTLU, he worked with University of Kent, Canterbury as postdoc 2017-2019, and University of Colorado, Colorado Springs as research associate 2019-2021. He received the Ph.D. degree in telematics systems engineering from Technical University of Madrid (UPM) in 2017. His research interests include cybersecurity, cloud computing, network softwarization, blockchain, and machine learning. He is a member of the IEEE.

Hsiang-Jen Hong received the Ph.D. degree from the National Taiwan University of Science and Technology, Taiwan, in 2018. He is currently a post-doctoral associate at the University of Colorado, Colorado Springs. His research interests are services computing, applied cryptography, and combinatorial optimization.

Jinoh Kim received his Ph.D. degree in Computer Science from University of Minnesota, Twin Cities. He is currently an Associate Professor of Computer Science at Texas A&M University-Commerce. His main research interest lies in the area of networked systems with the focuses on performance, reliability, scalability, visibility, and security.

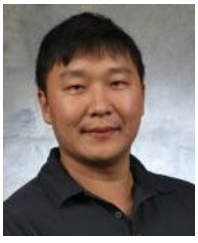


Simeon Wuthier is a Ph.D. student from the Department of Computer Science at the University of Colorado, Colorado Springs. His research is in theoretical computer science, cryptography, and distributed ledger technology.

Makiya Nakashima received his Master's degree in Computer Science from Texas A&M University-Commerce in 2020. While obtaining his Master's degree, he was a graduate research assistant at Texas A&M University-Commerce for 2019-2020.He is currently a bioinformatics technician at Cleveland Clinic Foundation. His research interests are in image analysis, computer vision and machine learning.

Xiaobo Zhou obtained the BS, MS, and PhD degrees in Computer Science from Nanjing University, in 1994, 1997, and 2000, respectively. Currently he is a professor of the Department of Computer Science, University of Colorado, Colorado Springs. His research lies in Cloud computing and datacenters, BigData parallel and distributed processing, autonomic and sustainable computing. He was a recipient of the NSF CAREER Award in 2009. He is a senior member of the IEEE.

Ching-Hua Edward Chow is Professor of Computer Science at the University of Colorado Colorado Springs. He got his PhD in Computer Science Degree from University of Texas at Austin 1985. He served as a Member of Technical Staff with Bell Communications Research 1986-1991. His research is focused on the improvement of the security, reliability and performance of network systems.



Sang-Yoon Chang (M'14) received the B.S. and Ph.D. degrees from the Department of Electrical and Computer Engineering at University of Illinois at Urbana-Champaign in 2007 and 2013, respectively. He is an assistant professor at the Computer Science Department at University of Colorado Colorado Springs. He was a post-doctoral fellow with the Advanced Digital Sciences Center. His research is in security, networking, wireless, cyber-physical systems, and applied cryptography.