Anomaly Detection based on Traffic Monitoring for Secure Blockchain Networking

Jinoh Kim* Makiya Nakashima* Wenjun Fan† Simeon Wuthier† Xiaobo Zhou† Ikkyun Kim‡ Sang-Yoon Chang†
Computer Science Department, Texas A&M University, Commerce, TX 75428, USA *
Department of Computer Science, University of Colorado, Colorado Springs, CO 80918, USA †
Cybersecurity Research Division, ETRI, Daejeon 34129, Korea ‡

***Abstract*—While the blockchain technology provides strong cryptographic protection on the ledger and the system operations, the underlying blockchain networking remains vulnerable due to potential threats such as denial of service (DoS), Eclipse, spoofing, and Sybil attacks. Effectively detecting such malicious events should thus be an essential task for securing blockchain networks and services. Due to its importance, several studies investigated anomaly detection in Bitcoin and blockchain networks, but their analyses mainly focused on the blockchain ledger in the application context (e.g., transactions) and targets specific types of attacks (e.g., double-spending, deanonymization, etc). In this study, we present a security mechanism based on the analysis of *blockchain network traffic statistics* (rather than ledger data) to detect malicious events, through the functions of *data collection* and *anomaly detection*. The data collection engine senses the underlying blockchain traffic and generates multi-dimensional data streams in a periodic manner. The anomaly detection engine then detects anomalies from the created data instances based on semi-supervised learning, which is capable of detecting previously *unseen* patterns, and we introduce our profiling-based detection engine implemented on top of AutoEncoder (AE). Our experimental results support the effectiveness of the presented security mechanism for accurate, online detection of malicious events from blockchain networking traffic data. We also show further reduction in time complexity (up to 66.8% for training and 85.7% for testing), without any performance degradation using feature prioritization compared to the utilization of the entire features.**

***Index Terms*—Blockchain, Bitcoin, P2P networking, traffic analysis, anomaly detection, machine learning, semi-supervised learning, online detection**

I. Introduction

The blockchain technology is widely applied for many application domains, including Internet of Things (IoT), transportation, software engineering, and financial technology [1], [2]. In particular, digital currencies such as Bitcoin [3] and Ethereum [4] build on a distributed blockchain for managing transaction records, with the provision of computational integrity through strong hash protection. A critical security hole is, however, the underlying blockchain networking based on peer-to-peer (P2P) communications that may not be protected by the computational cryptography. For instance, the distributed consensus protocol, which is the core operation in Bitcoin, takes place with little integrity and authenticity support (e.g., in the form of cleartext) over P2P overlays. Previous studies have also shown not a few vulnerabilities of blockchain infrastructures against cyberattacks, such as distributed denial of service (DDoS) [5], Eclipse [6], [7], spoofing [8], and Sybil attacks [9].

Anomaly detection has long been an active research topic to keep track of the health of the system as the second line of defense tool [10]. This function is essential to make timely responses by detecting malicious actions if present. Unlike the traditional rule-based detection that is limited to *known* attacks due to the reliance on pre-defined signatures, anomaly detection would be capable of discriminating previously *unseen* attacks from permitted actions by characterizing legitimate activity patterns. With its importance, several studies investigated anomaly detection in Bitcoin and blockchain networks, but their analysis mainly focused on the blockchain ledger in the application context (e.g., transactions) and targets specific types of attacks (e.g., double-spending, deanonymization, etc) [11]–[15]. In this study, we take a different approach based on the measurement and analysis of blockchain network traffic traces (rather than ledger data) to detect malicious events for securing blockchain networks. Our work is also different from the traditional network anomaly detection relying on connection data (e.g., KDDCup 1999 [16] and UNSW-NB15 [17]), since what we are interested in is *not* whether a connection between a pair of nodes is anomalous, *but* whether the traffic exchanged between peer nodes contains anything harmful. This is because the connection between two blockchain nodes is regarded as an *overlay link* used to exchange application data (e.g., for broadcasting Bitcoin messages).

In this paper, we present a security mechanism that offers accurate, online anomaly detection from blockchain network traffic data. Our security mechanism consists of two main components of *data collection* and *anomaly detection*. The data collection engine senses the underlying blockchain traffic and generates data streams to be analyzed in a periodic manner (e.g., one sample per second). We define a set of features to capture the blockchain traffic characteristics (e.g., number of packets/bytes and their differences over time), which facilitates data-driven analysis. We introduce our prototype system that creates data instances by sensing blockchain traffic from the real-world Bitcoin network (Mainnet), with a collection of datasets for both normal and attack vectors used for the learning and evaluation.

For designing the anomaly detection engine, our primary considerations are placed on *accuracy* and *time complexity*. We first aim to design an anomaly detection model yielding sufficiently high detection rates while minimizing false positives. Time complexity is a crucial concern for greater scalability with minimal resource usage. This is particularly critical for the testing (as opposed to the training that can be done beforehand with plenty of computing resources often with a relaxed time constraint) to enable real-time detection of threats, which is a key requirement to make timely responses that protect the system. With these considerations, we take a *semi-supervised learning* approach based on the profiling of normal activities, which is practical compared to a supervised learning approach that requires the collection of a wide variety of attack vectors with laborious tasks for labeling data with in-depth domain knowledge. Our anomaly detection scheme also contains *feature prioritization* to optimize training and testing time complexities without any significant performance loss.

Contributions. The key contributions of this paper can be summarized as follows:

- We introduce our data collection scheme with the implemented prototype system that collects the traffic from the real-world Bitcoin network (Mainnet). The collected traffic is sanitized and transformed into a data instance, and we define a set of features to represent traffic characteristics for a given time interval (Table I). We also collect simulated attack traffic generated in an isolated setting disconnected from the regular service for the evaluation.
- We present our anomaly detection scheme based on a semi-supervised learning approach with the capability for detecting previously unseen patterns through one-class learning. We introduce our detection engine implemented on top of AutoEncoder (AE) for profiling background patterns. The anomaly detection procedure is also discussed with the description of the core functions.
- We evaluate the presented security mechanism using both real and simulated traffic data and report our experimental results with the metrics of detection performance and time complexity. We also examine the impact of packet counting and bandwidth features, the result of which shows the approximate detection performance with a signification reduction of time complexity. A comparison study is conducted with other ML models (including both supervised and semi-supervised) to validate the effectiveness of our design choice based on the AE structure.

The organization of this paper is as follows. Section II provides the background of blockchain networking with the security challenges and a summary of related studies, In Section III, we introduce the prototype system for data measurement and the detailed description of the dataset are introduced, and Section IV presents our anomaly detection scheme based on AE with the description of the procedure, and Section V evaluates the proposed scheme in diverse settings with the comparison study with other ML-based detection techniques. We conclude our presentation with future directions in Section VI.

II. Background and Related Work

Digital Currency such as Bitcoin builds on a distributed blockchain to forgo a centralized third party (the bank) in storing and processing financial transactions. Blockchain secures the integrity of financial transactions by using the digital signature to track the owner account of the currency and by using the hash function to build dependency/chain across the blocks forming an immutable ledger. In the meantime, the underlying blockchain networking based on P2P communications lacks the security protection of the source node integrity or authentication because it is permissionless where there is no control in trust and identity registration (which can otherwise be used to control and limit the participation of P2P nodes). Due to the lack of protection, digital currency blockchain networking is vulnerable to attacks, such as DDoS attack [18], Eclipse attack [6], and double-spending attack [19]. In particular, Eclipse provides the attacker the capability of completely controlling the victim's view of the blockchain network and of forgoing the application-layer mechanisms to protect transaction integrity (e.g., double spending and selfish mining). Our work is motivated with the lack of protection for blockchain networking and designs an anomaly detection method as a means to secure the underlying Bitcoin networking.

There have been several studies investigated data analysis and detection problems in the Bitcoin environment using ML and statistical approaches and we next provide a summary of the previous studies closely related to our work.

Un-/Semi-supervised ML for Blockchain. Hirshman et al. [20] utilized clustering to detect anomalous behaviors from the Bitcoin transaction data in an unsupervised manner. Baqer et al. [12] also applied k-means clustering to detect spam transactions for DoS attacks on Bitcoin. These unsupervised techniques would not be suitable for online detection as clustering often assumes a batch processing after collecting a sufficient amount of transaction records. The work of squirRL [21] used a deep reinforcement learning approach to detect attack activities in blockchain but focused on incentive mechanisms, including selfish mining and block withholding attacks in Bitcoin. The study in [22] employed a semi-supervised learning approach using One Class-Support Vector Machine (OC-SVM) for anomaly detection in the Bitcoin network. While closely related to our work, the focus of this previous work is to detect suspicious nodes and transactions by analyzing Bitcoin transaction data (rather than analyzing blockchain traffic to detect potential attacks). Our experimental result also shows that OC-SVM would not be a feasible option with unacceptable performance and highly expensive time complexities for both training and testing, as will be discussed in Section V.

Supervised ML for Blockchain. Another class of studies employed supervised ML techniques for blockchain anomaly detection. Yin and Vatrapu [13] studied cyber crimes in the context of Bitcoin transaction data and evaluated a set of conventional supervised ML classifiers. Harlev et al. [23] discovered unidentified entities on the Bitcoin network using supervised ML classifiers. Tang et al. [14] presented a deep learning approach for classifying behavior patterns which are defined by the amount of the sequence data of transaction between peers. Sayadi et al. [15] used SVM to detect anomalies in Bitcoin transactions and also applied the k-means clustering to group similar outliers. In this work, we are more interested in a semi-supervised approach since supervised methods require both normal and attack instances for creating ML models, which may be infeasible in practice to collect samples for a wide variety of attack vectors (and it is impossible to obtain samples for emerging ones).

Statistical Analysis for Blockchain. Several studies conducted statistical analysis on Bitcoin transaction traffic. Ron and Shamir [24] performed analysis on the statistical properties of the Bitcoin transactional graph. Koshy et al. [25] collected the real-time transaction traffic and analyzed the data to create the mapping between Bitcoin addresses to IP addresses. Neudecker and Hartenstein [26] analyzed the P2P network traffic to see whether it is useful in the deanonymization of Bitcoin users. Biryukov and Tikhomirov [27] focused on the propagation timing information and clustering-based mechanism using such information for the Bitcoin user deanonymization. The main focus in these studies is on deanonymizing Bitcoin users through the statistical analysis on Bitcoin transaction traffic.

Overall, previous studies rely on the chained block data (e.g., transaction ledgers) with the knowledge of the application context to detect specific types of attacks. Our work in this paper focuses on blockchain networking traffic and detects malicious events through the analysis of the statistical information characterized from the underlying traffic data.

III. Data Collection and Prototyping

The data collection plays an essential role to facilitate the anomaly detection process by capturing statistical characteristics of blockchain traffic. In this paper, we focus on the most typical digital currency of Bitcoin to prototype and evaluate our security mechanism for anomaly detection. Nevertheless, our anomaly detection engine is generally applicable to any P2P-based blockchain networking applications where the peer relays and communicates to other fellow peers. In this section, we first present our Bitcoin core node implementation for data collection, and then provide a summary of our dataset with the description of the features defined.

A. Bitcoin Node Prototype

An active Bitcoin node is implemented for data collection using the Bitcoin core setup (software version Satoshi 0.18.0 and protocol version 70015). The Bitcoin core implementation has 26 Bitcoin message types for carrying out communication between peers. The Bitcoin node is assigned by a private IP address and maintains up to eight peer connections (which is configurable). The node initiates those connections (i.e. outbound connections) to the peers on the Bitcoin Mainnet as a private node. The Bitcoin node hosts our anomaly detection engine, as well as performing data collection.

We also implement another node ("simulated node") for simulating attacks, which connects to our Bitcoin node but never connects to the Bitcoin Mainnet. Thus, the simulated attacks will never exfiltrate from our testnet to the public network. More specifically, the Bitcoin node is connected to both the peer nodes from the Bitcoin Mainnet (outside of our control) and the simulated node we control for generating the attacking traffic. For this, our Bitcoin node has one connection to the simulated node and the rest of the connections to the Bitcoin Mainnet.

TABLE I
DESCRIPTION OF FEATURES (FEATURE 2–11 ARE COLLECTED FOR EACH MESSAGE TYPE.)

Index	Feature	Description
1	ClocksPerSec	The clock cycles cost per second by the node
2	# m	The number of m messages received in all
3	# m Diff	The number of m messages received for current logging entry
4	# m ClocksSum Diff	The aggregate clocks cost for processing the messages of m for current logging entry
5	# m ClocksAvg Diff	The average clocks cost for processing the messages of m for current logging entry
6	m ClocksAvg	The average clocks cost for processing the messages of m
7	m ClocksMax	The maximum clocks cost for processing the messages of m
8	m BytesSum Diff	The aggregate Byte size of the messages of m for current logging entry
9	m BytesAvg Diff	The average Byte size of the messages of m for current logging entry
10	m BytesAvg	The average Byte size of the messages of m
11	m BytesMax	The maximum Byte size of the messages of m

B. Data Collection and Feature Extraction

The Bitcoin node described above senses blockchain traffic from the peer connections and generates the relevant statistical information in a periodic manner. Since a fine-grain collection in a too short time interval may be unstable with potential inaccuracies, we set the collection interval to one second by default (i.e., one instance per second).

We define a set of features to represent the captured traffic. Our data collection is not based on TCP connections but is the aggregated statistical traffic information within the predefined time interval. Table I shows a list of features defined in this study to characterize blockchain traffic. In the table, feature #1 is common, while the other features (from #2 to #11) are specific to each message type. We use $m \in M$ where M is the set of 26 Bitcoin message types[1]. For example, $m = \texttt{VERSION}$ while $M = \{\texttt{VERSION}, \texttt{INV}, \texttt{TX}, \texttt{SENDCMPCT}, ...\}$. Since ten features are defined for each message type, there exist 260 features for the entire message types in total. As can be seen from the table, the defined feature set includes packet counting and bandwidth variables in addition to clock-related variables.

For actual data collection, the samples for the *Normal* dataset were collected by our prototype node through its natural connections to the Bitcoin Mainnet, which was then sanitized by manual inspection. We do not attempt to distinguish between the peers (those providing large traffic vs.

[1]https://developer.bitcoin.org/reference/p2p_networking.html

TABLE II
DATASET DESCRIPTION

Dataset	Description	Threat
Normal	Regular networking from Bitcoin Mainnet	None (sanitized)
Abn-1p	Abnormal with only 1 peer connection	Eclipse
Abn-Syn	Abnormal with continuous synchronization	Eclipse
Abn-DoS	Abnormal under flooding-based DoS	DoS

the others with lower networking traffic) and do not artificially select the peers (e.g., to filter those which do not relay as much). Therefore, there is no guarantee that all the peers connected to the prototype node are well-connected. Our approach, however, is more representative of the normal connections made by a Bitcoin node on Mainnet using random peer selections and our prototype node connects to the Mainnet using the default setting. To provide greater samples to smooth out the randomness across the connected peers, we re-select and re-connect using the Bitcoin-default random peer selection every 50 minutes or every five block arrivals in expectation.

We also collect attack traffic for DoS and Eclipse for the purpose of evaluation using our simulated node. The simulated flooding traffic provides the Abnormal-DoS dataset (*Abn-DoS*). The Eclipse attack simulation generates the other two abnormal cases. The Eclipse attacker simply limiting the peer/networking diversity of the victim but otherwise 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 above mentioned datasets as well as their related threats are described in Table II. Finally, the total number of data instances collected in this study is 639,360 entries (i.e., spanning over 7.4 days), and 39.4% of the data belong to *Normal* (see Table III).

In this work, we focus on Bitcoin implementation for prototyping and experiments. Nonetheless, the work is generally applicable to permissionless P2P networking applications including other cryptocurrencies in principle, as our traffic analyses use the standard networking parameters/features and only distinguish the message types. Targeting other cryptocurrency implementations such as Ethereum would require fine-tuning the features (while building on most of the generic P2P-networking features) and re-training the model so that it matches the base implementation and the Normal profile based on the targeted implementation. More implementation-specific studies, including identifying especially potent features for secure anomaly detection and exploring application-layer deep-packet inspection approaches, are left for future work.

IV. Anomaly Detection Scheme

A. Our Scheme Approach

This paper tackles the detection of anomalies in a permissionless blockchain network for digital currencies using an ML approach. Semi-supervised learning constructs the learning model based on the profiling of background characteristics [10]. To design anomaly detection, therefore, the semi-supervised approach is advantageous since it is less complicated to collect instances for modeling legitimate patterns, compared to ones for malicious events given a wide variety of attack vectors that may be dynamically evolving, including previously unseen zero-day networking exploits. We take the semi-supervised learning approach to realize our anomaly detection engine in this study.

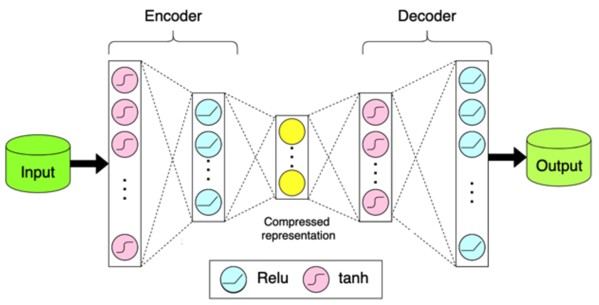


Fig. 1. AutoEncoder structure: AE consists of the encoder compressing the input and the decoder restoring the compressed representation. This structure shows a mixture of `ReLU` and `tanh` activation units as an example.

One-class Support Vector Machine (OC-SVM) and AE (and its variants such as Variational AutoEncoder) are well-known ML algorithms that have been utilized for implementing semi-supervised learning. As its name suggests, OC-SVM is a variant of Support Vector Machine (SVM). Despite its popular use, a potential weakness of OC-SVM is that it is sometimes working unstably if the dataset contains outliers and multi-dimensional features [28], [29]. Moreover, its complexity is substantial, which would be a critical obstacle to employ it for time-sensitive operations. Hence, it may not be a good candidate to analyze our dataset, which is complex with high dimensionality, with the real-time detection requirement. On the other hand, AutoEncoder (AE) is based on a neural network structure, with an encoder that learns a representation of data for dimension reduction and a decoder that learns how to reconstruct the input with the minimized reconstruction error [29], [30]. Our comparison study shows that AE is more accurate and much lighter than OC-SVM for both learning and testing (as will be discussed in Section V-D), and we develop our anomaly detection function on top of AE.

There exist several different variants of AE, such as denoising AE (DAE), convolutional AE (CAE), and variational AE (VAE), which have been considered for traditional network anomaly detection [30]–[32]. For instance, VAE has been adopted in several studies [30], [31], while some of recent studies utilized the vanilla AE for the same purpose [33]–[35]. VAE is known as a powerful generative method by using a probabilistic model such as Guassian distribution for internal representation. VAE relies on reconstruction probability (rather than reconstruction error), which indicates the probability that the instance in question originated from the trained distribution. This probability information is indeed helpful for scoring anomalies as a quantitative measure. But if the detection process performs classification based on the

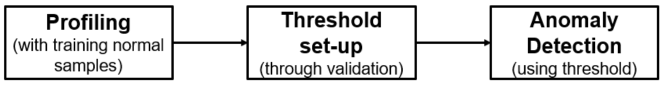


Fig. 2. Anomaly detection procedure: (1) *Profiling* constructs a learning model from normal samples; (2) *Threshold set-up* discovers a threshold value to discriminate anomalies; and (3) *Anomaly detection* tests the data point in question to determine whether it is normal using the threshold.

reconstruction probability, a data instance lower than $x\%$ from the probability density (e.g., $<5\%$) would be regarded as an anomaly as previous studies assumed, which results in approximately a $x\%$ false positive rate if the model is trained with normal samples. A high rate of false positives is critical in practice considering the base-rate fallacy. Note that we train the detection model using the sanitized normal samples as described in Section III-A. With the standard AE, setting up the threshold to discriminate anomalies from normal would be relatively straightforward based on the distribution of the reconstruction errors measured with the training samples [35]. Any data instance showing a negligible error lower than the threshold will then be marked as 'normal' and vice versa. In this study, we utilize AE to design our detection engine.

B. Anomaly Detection using AutoEncoder

We introduce the standard AE structure as the foundational element of our scheme, and present our anomaly detection procedure in detail with the core functions.

Fig. 1 illustrates an example of a stacked AE structure composed of the encoder and the decoder with multiple hidden layers internally. The encoder compresses the input data, transforming input (x) into a compressed representation (x'): $x \rightarrow x'$ and dim(x) $>$ dim(x'), where dim(v) stands for the dimension of variable v. The decoder performs the reversal operation and restores the compressed representation (x') to an expanded representation (x''): $x' \rightarrow x''$ and dim(x) == dim(x''). Here, reconstruction error (ϵ) is defined as the difference between the input and the restored representation: $\epsilon = |x - x''|$. Hence, $\epsilon = 0$ indicates a loss-less compression.

The learning process performs in a way to decrease the reconstruction error in the profiling time. The activation function is a component that determines the output of a neural network, and `ReLU` and `tanh` are well-known units for activating [36]. The figure shows the use of `ReLU` and `tanh` for the activation function as an example. There can exist different settings to set up a learning model by configuring hyperparameters, and we will discuss the configuration of AE used for our experiments, shortly with Table IV in Section V. Finally, AE reconstruction errors calculated from normal samples can be used to perform anomaly detection; that is, certain data points having relatively high errors could be assumed that it is *not* from the normal class.

C. Anomaly Detection Procedure

We next describe the details of the anomaly detection procedure based on AE. Fig. 2 summarizes the procedure of anomaly detection, consisting of three stages, as follows:

TABLE III
NUMBER OF INSTANCES IN TRAINING AND TESTING FILES

	Normal	*Abn_1p*	*Abn_DoS*	*Abn_Syn*
Training (80%)	201,600	124,356	130,412	55,120
Testing (20%)	50,400	31,089	32,603	13,780

TABLE IV
AE MODEL PARAMETER SETTING

Parameter	Setting
No. layers	{2, 4, 6, 8}
Activation	{`ReLU`, `tanh`, `Mix` (of `ReLU` & `tanh`)}
Loss	default (MSE)
Learning rate	default (0.001)
Epochs	No. layers × 25
Batch size	100
Optimizer	Adam

1) *Profiling*: This stage fully utilizes the AE learning process, and a learning model is created from normal samples provided for training. We will examine different AE settings by applying different parameters.
2) *Threshold set-up*: In this stage, the density of reconstruction errors is analyzed by re-applying the training samples to the learned model (*validation*), in order to discover a relevant value for threshold (τ). A set of AE models based on different configuration parameters are compared based on the resulted distributions. We will investigate the impact of threshold settings on detection performance.
3) *Anomaly detection*: This stage actually tests the data point in question using the threshold discovered in the previous stage. A commonly accepted assumption is that reconstruction errors for normal instances are relatively low compared to anomalies. For a data point d_i, the reconstruction error ϵ_i is calculated by using the constructed learning model (in *Profiling*). Then the anomaly detection function determines d_i is normal if $\epsilon_i < \tau$; anomalous, otherwise.

V. EVALUATION

In this section, we first describe our experimental setting, and then report the experimental results with the metrics of detection performance and time complexity. We also examine the impact of packet counting and bandwidth features, which would be crucial for optimizing time complexity. The proposed AE model will be compared to other candidate ML models based on semi-supervised and supervised approaches.

A. Experimental Setting

The experiments were conducted on a job submission-based HPC system. The CPU type in the system is Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz. The system allocates 64 cores to individual jobs by default.

The dataset is partitioned for training and testing: first 80% of the dataset for training (to build a learning model), and the rest 20% of the dataset for testing (to classify one into either normal or anomaly). Table III shows the composition of training and testing datasets with the ratio of 80%:20%.

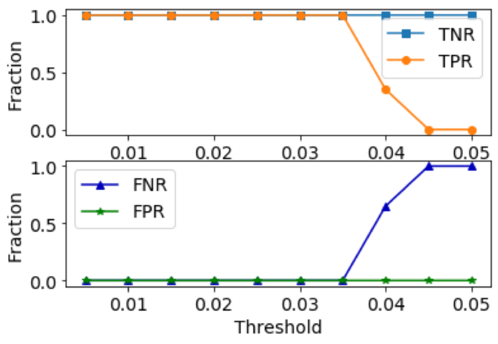


Fig. 3. Performance details over threshold (for the model with two layers and `tanh`): TNR and FPR show the perfect result (i.e., TNR=100% and FPR=0%) across the threshold range, whereas TPR and FNR cross over at around $\tau = 0.04$ due to the incorrect classification of anomalies to normal.

For semi-supervised learning, profiling is conducted with the normal data only, which means that attack instances in the training set are not used in the learning phase. In contrast, supervised learning relies on the entire samples in the training set (including both normal and attack instances) to create a learning model.

To measure detection performance, we use the standard metrics of *accuracy* and *F1 score* defined: Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$ and F1 score $= \frac{2TP}{2TP+FP+FN}$, where TP=true positives, FP=false positives, FN =false negatives, and TN=true negatives. Accuracy is widely used but could be biased if the population of the minority class (i.e., either normal or anomaly) is too small. In contrast, F1 score is known to be more reliable to the class imbalance problem. As can be seen from Table III, our dataset is not significantly biased and we observed mutually agreed results by the two metrics. For measuring time complexity, the same experiment is executed twice with an interval of two weeks and the smallest cost is selected to be reported.

To configure ML parameters, we examined a set of AE settings with different parameter values in our preliminary experiment. Table IV shows the parameter space considered for our experiments. We observed that `tanh` works slightly better, while configuring a greater number of layers is not helpful to lower reconstruction errors but increases the training time sub-linearly. From the initial observation, we set activation=`tanh` and # layers=2 as the default configuration. Additionally, the number of neurons for each layer is simply chosen in proportion to the number of features in the dataset ($dim(F)$, where F is the input feature set), based on the following rule: the first layer for encoder = $dim(F)/2$, the second and subsequent = $dim(F)/3$, the size of the compressed representation = $dim(F)/4$, and vice versa for decoder (i.e., symmetric). Although more a rigorous examination would help discover optimal configuration settings, our interest in this study is not in the optimization of ML models and we leave it as one of future tasks.

B. Anomaly Detection Performance

We now report the detection performance of the proposed AE-based anomaly detection model. Fig. 3 shows true positive rate (TPR; actually anomaly and classified to anomaly), true negative rate (TNR; actually normal and classified to normal), false positive rate (FPR; actually normal but misclassified to anomaly), and false negative rate (FNR; actually anomaly but misclassified into normal) when using the model configured with two layers with the `tanh` unit. The figure shows the perfect result for TNR and FPR (i.e., TNR=100% and FPR=0%) across the threshold range, which confirms the entire normal instances are classified correctly. However, TPR and FNR cross over at $\tau = 0.04$ because certain anomalies (having relatively low errors) are incorrectly classified and assumed as normal. The result indicates the available value range for the threshold is not narrow and tight.

By looking at the break-down of the accuracy based on individual classes in Fig. 4. The normal instances (*Normal*) is always correctly classified, while the attack classes show the degraded performance as we choose a greater value to set the threshold. The *Abn-1p* instances begin to be misclassified when $\tau > 0.04$. To see at what point the other two classes (*Abn-Syn* and *Abn-DoS*) become misclassified, we extend the threshold setting to 0.1. The figure shows that reconstruction errors for those two attack classes are considerably large. This indicates *Abn-1p* instances are closer to *Normal* than the instances belonging to the other two attack classes.

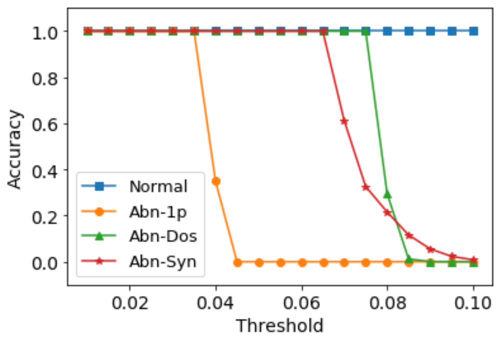


Fig. 4. Detection accuracy for individual classes (using the model with two layers and `tanh`)

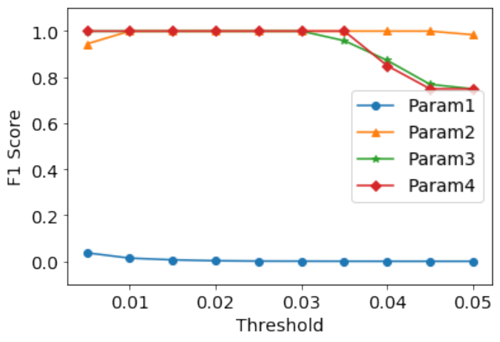


Fig. 5. Impact of counting and bandwidth features (# layers = 2 and activation=`tanh`): *Param 3* shows highly consistent results, while *Param 1* utilizing a too small number of features shows unacceptable performance. *Param 2* works better than *Param 1* but shows some degraded performance at the early stage.

C. Impact of Counting/Bandwidth Features

We are interested in investigating the impact of the features related to packet counting and bandwidth with their wide use in network security and traffic analysis [37], [38]. Hence, this investigation is connected to feature selection that discovers relatively more important features [39]–[41]. If successful, the learning performance resulted with the chosen feature subset would approximate to the original (observed without performing reduction) or even better with a reduced time complexity for training and testing. In this study, we focus on the following three variables related to the packet counting and bandwidth: $\#m$ (Index=2), $\#m$ Diff (Index=3), m BytesAvgDiff (Index=9), where the index comes from Table I. Since there are 26 usable message types (as described in Section III), the use of a single variable means the utilization of 26 features (i.e., one feature for each message type). We compose the following feature groups:

- *Param 1*: Use a single variable of $\#m$ for each message type (26 features in total; 26/261≈10%)
- *Param 2*: Use two variables of $\#m$ and $\#m$ Diff for each message type (52 features in total; 52/261≈20%)
- *Param 3*: Use all three variables for each message type (78 features in total; 78/261≈30%)
- *Param 4*: No reduction applied (default=100%)

Fig. 5 shows the impact of the counting and bandwidth features on detection performance. As can be seen from the figure, utilizing a too small number of features shows completely unacceptable performance in *Param 1*, while *Param 2* works better but shows some degraded performance at the early stage. From the figure, *Param 3* works well consistently when $\tau \leq 0.03$. Again, *Param 3* defines a subset of features related to packet counting and bandwidth variables widely used in the networking traffic analyses. In fact, *Param 3* filters out some redundant features that can be derived from other features in its set. By referencing a smaller number of features, *Param 3* can reduce time complexity for executing the detection function without any performance loss, compared to *Param 4* relying on the entire features defined in Section III, as will be discussed shortly.

D. Comparison with Other ML Methods

We compare our proposed scheme with other ML-based anomaly detection techniques. As mentioned, we consider OC-SVM for profiling normal behaviors with its popular use for that purpose. We also examine conventional supervised learning techniques widely utilized for anomaly detection. The supervised learning approach could be advantageous in the presence of both normal and attack instances with the associated label information in the training dataset. However, obtaining samples for a wide variety of threats is *not* a trivial task requiring arduous manual work to simulate such attacks in a highly controlled setting by skilled experts. Even more critically, supervised learning has an intrinsic weakness of the detection of unseen attack classes (due to the lack of the adequate information to discriminate them in the learning model).

TABLE V
PER-INSTANCE TRAINING AND TESTING COMPLEXITIES

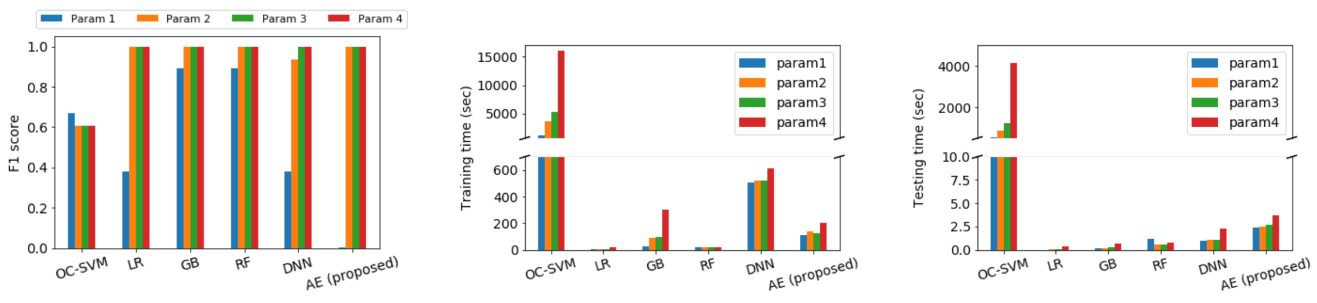
ML algorithm	Training cost ($msec$)		Testing cost ($usec$)	
	Param 4	*Param 3*	*Param 4*	*Param 3*
OC-SVM	79.9	26.5	32445.7	9782.4
LR	0.029	0.015	3.14	0.45
GB	0.59	0.19	5.38	2.28
RF	0.036	0.036	6.25	4.28
DNN	1.20	1.03	17.5	8.74
AE (proposed)	1.00	0.63	28.7	20.8

The ML algorithms considered in this comparison study include One Class Support Vector Machine (OC-SVM), Logistic Regression (LR), Random Forest (RF), Gradient Boosting (GB), and Deep Neural Network (DNN). The detailed description of the algorithms can be found from [42]. Again, we basically utilize the default setting without intensive optimizations for individual ML models, since the optimization is not one of the main interests in this study. Here is a brief description of the configuration setting. We utilize LR and OC-SVM with no further specific settings. For GB and RF, we used 100 trees, respectively. The hyperparameter configuration for DNN is as follows: 4 hidden layers with `ReLU` for activation, `sigmoid` for output layer activation, 0.2 for dropout, 50 for epochs, 0.001 for learning rate, binary cross-entropy for loss, 100 for batch size, and Adam optimizer.

Fig. 6 compares the ML models including our proposed detection model, with respect to detection performance in F1 score, training time, and testing time when using different feature groups (from *Param 1* to *Param 4*). As can be seen from the figure, OC-SVM is not a good option showing poor performance (less than 70% F1 score). Largely, the supervised learning models perform very well if the feature set used is not too small (i.e., other than *Param 1*), showing almost perfect detection performance when using *Param3* and *Param 4*. Our AE model is configured with two layers, `tanh` for activation, and threshold $\tau = 0.02$, and it shows comparable performance consistently when using any feature sets other than *Param 1*. Again, the ML models other than OC-SVM and AE are based on the supervised learning approach that requires both normal and anomalous samples for training with the associated label information.

The training and testing complexities incur overheads in anomaly detection, and hence, managing those complexities is a crucial task for scalable processing in data analytics. In particular, the testing time is critical to realize the real-time detection. Fig. 6(b) and Fig. 6(c) show the time taken for training and testing, respectively. OC-SVM is hugely expensive for both training and testing. The training cost for AE is cheaper than DNN and comparable to GB. The testing cost for AE is slightly larger than ones for the supervised models. Again, AE refers to the normal samples for training (that includes both profiling and validation), while supervised learning needs the normal and attack samples for training (which includes learning only without validation); the equal number of data points are tested in the testing phase.

For further analysis, Table V provides the per-instance



(a) F1 score (b) Training time (c) Testing time

Fig. 6. Comparison of anomaly detection performance and time complexity: (a) AE is comparable to the supervised ML models, while OC-SVM shows poor detection performance; (b) The training cost for AE is cheaper than DNN and comparable to GB; (c) The testing cost for AE is slightly higher than the supervised models but significantly lower than the cost for OC-SVM.

analysis required for training and testing. As the training cost is generally much more expensive, the metric of training time reported in the table is milliseconds (msec) and testing time is in microseconds (usec). The table also shows the cost when referencing the entire features (*Param 4*) and a subset of features (*Param 3*) yielding the approximate detection performance. We can see that the training complexity is several orders of magnitude greater than the testing complexity. The max saving is 66.8% for training (with OC-SVM) and 85.7% for testing (with LR).

From Table V, the AE model requires 28.7 usec to test one instance. Given the use of 64 cores in our experiment, it may take less than 1.9 msec even on a machine with a single core for a single test. In case of *Param 3*, our detection scheme requires 52.8 usec for a single test (or <1.4 msec in case of the use of a single core). The cost range for testing, which scale is in msec or lower, is significantly lower than the sensing granularity (monitoring and logging every second) or the time-window size of the testing (processing and testing 20 minutes long data). The time cost of running our anomaly detection is therefore sufficiently cheap to perform real-time analysis. Our previous work analyzes the feasibility and cost in greater details with a focus on the ML detection algorithms' system impact on mining [43]. Further systems investigation to control the networking parameters, analyze the corresponding implementation costs, and compare them with the frequencies of the networking attacks and defense on Bitcoin or other blockchains are left for future work.

VI. Conclusions & Future Work

This paper presents a security mechanism capable of detecting anomalies from the underlying blockchain network traffic statistics focusing on cyberattacks on Bitcoin. We introduced our data collection scheme with the implemented prototype system that senses blockchain networking traffic and creates multi-dimensional data streams in a periodic manner. Our anomaly detection scheme is based on the semi-supervised learning approach that is practical and resistant to unseen threats through the profiling of background legitimate patterns, and we presented our detection engine based on AE as a realization of semi-supervised learning. We conducted experiments using the real dataset collected from the public Bitcoin network (Mainnet) with a set of simulated attack traces. The experimental results showed that our presented security mechanism is effective for accurate, online detection of malicious events from blockchain networking traffic data. In addition, our profiling-based detection engine implemented on top of AE yields comparable or even better performance for detecting anomalies than other candidate models based on semi-supervised (OC-SVM) and supervised (LR, GB, RF, and DNN) approaches. We also measured time complexity for the detection function, and the cost range for testing is at least an order of magnitude smaller than the sensing granularity, which enables real-time operations for detecting anomalies. Moreover, our experimental results showed a significant reduction of time complexity (up to 66.8% for training and 85.7% for testing), without any performance degradation using feature prioritization compared to the utilization of the entire features.

Our work aims to secure blockchain networking but focuses on the specific networking application of the blockchain-based digital currency in Bitcoin. While there is a surge of recent blockchain research to secure the application layer (including the cryptographic primitives and the distributed consensus protocol), the security study in the blockchain's underlying P2P network is relatively lacking and largely relies on those provided in the general networking context. We aim to address such gap and provide a Bitcoin-focused study to facilitate further research in securing the networking of blockchain. There are many potential future research directions to address the important problem, including the construction of the active security measures and responses after using our anomaly detection, the improvement of our anomaly detection scheme such as the ML algorithm improvement or optimization, and the systems investigation incorporating our scheme to the full operational flow of a blockchain peer node.

Acknowledgment

This research was supported in part by Colorado State Bill 18-086 and by the National Science Foundation under Grant No. 1922410. This work was also supported in part by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government [20ZR1300, Development of core technologies for trust data connectome to realize a safe data society].

References

[1] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018.

[2] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, 2017.

[3] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[4] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[5] A. Feder, N. Gandal, J. Hamrick, and T. Moore, "The impact of ddos and other security shocks on bitcoin currency exchanges: Evidence from mt. gox," *Journal of Cybersecurity*, vol. 3, no. 2, pp. 137–144, 2017.

[6] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proceedings of 24th USENIX Security Symposium (USENIX Security 15)*, pp. 129–144, Aug 2015.

[7] 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)*, pp. 305–320, 2016.

[8] J. R. v. d. Merwe, X. Zubizarreta, I. Lukčin, A. Rügamer, and W. Felber, "Classification of spoofing attack types," in *2018 European Navigation Conference (ENC)*, pp. 91–99, 2018.

[9] J. Dinger and H. Hartenstein, "Defending the sybil attack in p2p networks: taxonomy, challenges, and a proposal for self-registration," in *First International Conference on Availability, Reliability and Security (ARES'06)*, 2006.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[11] M. Signorini, M. Pontecorvi, W. Kanoun, and R. Di Pietro, "Bad: blockchain anomaly detection," *arXiv preprint arXiv:1807.03833*, 2018.

[12] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing out: Bitcoin "stress testing"," in *Proceedings of the International Conference on Financial Cryptography and Data Security(FC)*, pp. 3–18, 2016.

[13] 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)*, pp. 3690–3699, Dec 2017.

[14] 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, pp. 71208–71215, 2018.

[15] S. Sayadi, S. B. Rejeb, and Z. Choukair, "Anomaly detection model over blockchain electronic transactions," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 895–900, 2019.

[16] "KDD Cup 1999 Data." http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[17] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*, pp. 1–6, 2015.

[18] M. Vasek, M. Thornton, and T. Moore, "Empirical analysis of denial-of-service attacks in the bitcoin ecosystem," in *Proceedings of the International Conference on Financial Cryptography and Data Security(FC)*, pp. 57–71, 2014.

[19] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, "Misbehavior in bitcoin: A study of double-spending and accountability," *ACM Trans. Inf. Syst. Secur.*, vol. 18, pp. 2:1–2:32, May 2015.

[20] 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.

[21] 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.

[22] T. Pham and S. Lee, "Anomaly detection in bitcoin network using unsupervised learning methods," *arXiv preprint arXiv:1611.03941*, 2016.

[23] 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 *Proceedings of 51st Hawaii International Conference on System Sciences*, pp. 3497–3506, 2018.

[24] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Proceedings of the International Conference on Financial Cryptography and Data Security(FC)*, pp. 6–24, 2013.

[25] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *Proceedings of the International Conference on Financial Cryptography and Data Security(FC)*, pp. 469–485, 2014.

[26] T. Neudecker and H. Hartenstein, "Could network information facilitate address clustering in bitcoin?," in *Proceedings of the International Conference on Financial Cryptography and Data Security(FC)*, pp. 155–169, 2017.

[27] A. Biryukov and S. Tikhomirov, "Transaction clustering using network traffic analysis for bitcoin and derived blockchains," in *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 204–209, 2019.

[28] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of ACM SIGKDD Workshop on Outlier Detection and Description*, pp. 8–15, 2013.

[29] Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[30] 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)*, pp. 91–99, IEEE, 2019.

[31] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[32] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE Access*, vol. 6, pp. 48231–48246, 2018.

[33] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, The Internet Society, 2018.

[34] Y. Zhong, W. Chen, Z. Wang, Y. Chen, K. Wang, Y. Li, X. Yin, X. Shi, J. Yang, and K. Li, "Helad: A novel network anomaly detection model based on heterogeneous ensemble learning," *Computer Networks*, vol. 169, p. 107049, 2020.

[35] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen, "An unsupervised deep learning model for early network traffic anomaly detection," *IEEE Access*, vol. 8, pp. 30387–30399, 2020.

[36] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Advances in neural information processing systems*, pp. 4939–4948, 2018.

[37] S. Feghhi and D. J. Leith, "A web traffic analysis attack using only timing information," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1747–1759, 2016.

[38] Y. Qin and D. Huang, "A statistical traffic pattern discovery system for manets," in *MILCOM 2009-2009 IEEE Military Communications Conference*, pp. 1–7, IEEE, 2009.

[39] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[40] S. Alelyani, J. Tang, and H. Liu, "Feature selection for clustering: A review," in *Data Clustering*, pp. 29–60, Chapman and Hall/CRC, 2018.

[41] A. Jović, K. Brkić, and N. Bogunović, "A review of feature selection methods with applications," in *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 1200–1205, Ieee, 2015.

[42] S. Raschka, *Python machine learning*. Packt Publishing Ltd, 2015.

[43] W. Fan, J. Kim, I. Kim, X. Zhou, and S.-Y. Chang, "Performance analyses for applying machine learning on bitcoin miners," *International Conference on Electronics, Information, and Communication (ICEIC)*, 2021.