Towards Learning-Based, Content-Agnostic Detection of Social Bot Traffic

Yebo Feng, Jun Li, Lei Jiao, and Xintao Wu

Abstract—With the fast-growing popularity of online social networks (OSNs), the security and privacy of OSN ecosystems becomes essential for the public. Among threats OSNs face, malicious social bots have become the most common and detrimental. They are often employed to violate users' privacy, distribute spam, and disturb the financial market, posing a compelling need for effective social bot detection solutions. Unlike traditional social bot detection approaches that have strict requirements on data sources (e.g., private payload information, social relationships, or activity histories), this paper proposes a method called BotFlowMon that relies only on content-agnostic flow-level data as input to identify OSN bot traffic. BotFlowMon introduces several new algorithms and techniques to classify social bot traffic from real OSN user traffic, including aggregating network flow records to obtain OSN transaction data, fusing transaction data to extract features and visualize flows, and an innovative density-valley-based clustering algorithm to subdivide each transaction into individual actions. The evaluation shows BotFlowMon can identify the traffic from social bots with a 96.1% accuracy, which, based on the worst case study on a testing machine, only takes no more than 0.71 seconds on average after it sees the traffic.

Index Terms—Online Social Network, Social bot, Sybil account, NetFlow, Machine learning, Data mining

1 Introduction

The past decades have witnessed a rapid expansion of online social networks (OSN). Facebook has achieved more than 2.196 billion active users around the world and Twitter has reached 336 million users [1]. Unfortunately, OSNs are increasingly threatened by software-controlled social bots [2] that impersonate real OSN users for troublesome or malicious purposes [3]. Even though not all social bots are malicious, as many are used for customer service and information dissemination, various attacks, abuses, and manipulations are based on social bots [4], such as infiltrating Twitter [5], launching spam campaigns [6], and performing financial fraud [7].

Existing approaches to detecting social bots need to utilize the social relationship structure or private content data from users' accounts, all of which can lead to privacy infringement and can only be executed by OSN providers. In this paper, we propose a new, *content-agnostic* social bot detection method called BotFlowMon. It takes network traffic flow information as input, which is NetFlow records [8] in this paper, to differentiate the social bot traffic from the real user traffic. As Internet/network service providers can easily collect NetFlow records, it is thus also convenient for them to deploy BotFlowMon, making social bot detection no longer dominated by OSN providers. Moreover, as NetFlow records are coarse-grained summaries of packet headers and contain *no* OSN content data from packet payload [9], BotFlowMon is also privacy-preserving.

BotFlowMon can detect social bot traffic accurately and quickly. It harnesses the power of machine learning on big data for the best efficacy in labeling social bot traffic versus real user traffic. BotFlowMon employs five modules:

1) The *preprocessing module* that filters out noises and irrelevant data from raw NetFlow records and extracts OSN-related traffic flows;
2) The *flow aggregation module* that transfers the NetFlow records into transaction-level datasets, making the characteristics of social bots more apparent for detection;
3) The *transaction fingerprint generation module* that, with a new data fusion technique, extracts features from transaction-level datasets, normalizes the values, and visualizes the flows;
4) The *transaction subdivision module* that employs a new, density-valley-based clustering algorithm to further divide each transaction into multiple actions, thus reducing training data volume and accelerating training;
5) The *machine learning & classification module* that uses the action-level data to construct a transaction-level social bot classification model with convolutional neural network (CNN) and multilayer perceptron (MLP).

BotFlowMon is the first work that leverages content-agnostic traffic flows to detect social bots. By embracing a suite of newly introduced techniques, BotFlowMon provides social bot detection at behavior level, which is a finer granularity than existing approaches; it can identify whether an individual OSN behavior is from a real user or a social bot account. It also can detect social bots in real time; as soon as it receives network traffic, BotFlowMon can start detection immediately.

Our evaluation with 535 gigabytes (GB) of raw NetFlow data from a large university environment shows that BotFlowMon can identify social bot traffic with an accuracy of 96.1% (or around 90% if the user adjusts the false positive rate to zero). Simultaneously, the conciseness of NetFlow data provides an advantage for fast and efficient data processing. Even only running on a laptop with 2.7-

- *Y. Feng, J. Li, and L. Jiao are with Computer & Info. Science, University of Oregon, Eugene, OR, 97403. E-mail: {yebof, lijun, jiao}@cs.uoregon.edu*
- *X. Wu is with Computer Science & Computer Engineering, University of Arkansas, Fayetteville, AR, 72701. E-mail: xintaowu@uark.edu*

Manuscript received October 31, 2019; revised June 27 and October 16, 2020.

Ghz CPU and 16-GB memory, BotFlowMon can support real-time social bot detection for a campus-level network; after the NetFlow records of a social bot are collected, it takes only 0.71 seconds on average to detect the social bot.

The rest of this paper is organized as follows. After we outline related work in Section 2, we describe BotFlowMon's design in Section 3 and evaluate it in Section 4. We discuss its limitations and open issues in Section 5 and conclude the paper in Section 6. Of a particular note here is that this paper is an extended version of work published in [10].

2 Related Work

As BotFlowMon is a content-agnostic approach to detecting social bots from network traffic, in this section we investigate other social bot detection approaches and content-agnostic traffic classification and anomaly detection work.

2.1 Social Bot Detection Approach

Various approaches have been developed to identify social bots. According to the input data, they are often either content-based or structure-based [23], with a new trend on using crowdsourcing techniques as well. Table 1 shows a general comparison between these methodologies. BotFlowMon is the only approach that only relies on network traffic.

2.1.1 Content-Based Approach

Content-based social bot detection approaches seek to detect behaviors, accounts, or account clusters associated with social bots using OSN content. Here, the OSN content refers to not only explicit information of an account, such as its profiles, URLs, and linguistic features of posts, but also implicit information of an account such as its clickstream, local graph structure, and user behavior. For example, research in [11], [24], [25] finds URLs in posts can help identify spam messages or social bot accounts; research in [12], [26], [27] leverages features related to account profiles or message content to determine if an account is actually a social bot; and research in [13], [28], [29] uses local graph structure information such as the number of followers and relationship between interactions for social bot detection.

Different methods have been used to model the differences of the OSN content between social bots and human users. While some used statistical analysis, including [12], [13], [26], [30], to be more flexible in tackling complicated features and more accurate, many leveraged machine learning techniques, such as random forest ([11], [28], [31]), SVM ([14], [27], [32]), and logistic regression ([24], [32]).

While often effective, content-based approaches have certain drawbacks. First, they require content data of OSN users. Compared with flow-level traffic data used in BotFlowMon, such data are usually privacy sensitive and could lead to potential privacy infringement. Second, they are vulnerable to adversarial attacks since social bot programs can mimic human users' behaviors to escape their detection.

2.1.2 Structure-Based Approach

The assumption of structure-based approaches is that social bot accounts (often referred as Sybil accounts in these approaches) can build connections between themselves arbitrarily, but it is difficult for them to establish or manipulate social relationship with human users [17]. The structural gap between social bot accounts and human accounts can then be used to identify social bots.

Structure-based approaches can be classified into two categories—random walk (RW) based and Markov Random Field (MRF) based. RW-based methods start walking from either a social bot node or a human user node and then use some classifiers to infer the labels of nodes along the path, as exemplified by research in [15], [33], [34]. MRF-based methods, such as those in [16], [18], [35], model the OSN structure as a Markov random field and use probability methods (e.g., Loopy Belief Propagation) to estimate each node's conditional probability of being a social bot. The two methods can be combined [36], [37].

Unlike content-based approaches, structure-based approaches are not vulnerable to content-oriented adversarial attacks. However, they have their own drawbacks: (1) Their aforementioned assumption is not always true. According to research in [38], it is easy for a social bot account to establish a relationship with a real user account. Moreover, once a real user account is compromised and becomes a social bot, it inherits all the social relationship of the real user. Consequently, structure-based approaches will have difficulty in detection social bots in such scenarios. BotFlowMon is not dependent on OSN structures, thus not subject to these issues. (2) They can only offer social bot detection at the account level, which is to determine whether an account is a real user or a social bot. BotFlowMon works at the behavior level which is a finer granularity. (3) They only detect social bot accounts with the change of social topology. If a social bot keeps posting messages without interacting with other accounts, they will fail to detect this social bot. In contrast, BotFlowMon detects social bots in real time soon after it receives NetFlow records of social bot traffic.

2.1.3 CrowdSourcing-Based Approach

A new trend in social bot detection is to utilize crowdsourcing [20], [21], [22], in which one can ask individual crowdworkers to judge whether a program is a bot or not and then aggregate the decisions from all crowdworkers. For example, research in [19] relies on a crowdsourcing layer to have individual users determine whether an OSN account is a bot account or not and a filtering layer to filter out unsatisfactory reports from individual users. Compared with BotFlowMon, a crowdsourcing approach tends to incur a high detection latency due to its interactions with human users, a high cost to pay crowdworkers, and privacy risks if distributing privacy-sensitive data to the crowd. Besides, a crowdsourcing approach usually detects social bots at account level, while BotFlowMon does so at behavior level.

2.2 Content-Agnostic Traffic Classification & Anomaly Detection

During the early development of traffic classification, content-agnostic traffic classifications was elementary and mainly focused on classifying traffic into a few frequently used applications, such as telnet, https, and BitTorrent. Different traffic data were used as input. For example, research in [39] used the size, inter-arrival time, and arrival order of IP packets; research in [40] leveraged packet header information; and research in [41] utilized metadata of packets,

TABLE 1
Comparisons of different social bot detection approaches.

Category	Representative work	Primary input	Detection granularity	Timeliness	Privacy preservation
Content-based	[11], [12], [13], [14], etc	OSN content	behavior, account (cluster)	mostly real time	No
Structure-based	[15], [16], [17], [18], etc	OSN topology	account	with the growth of topology	No
Crowdsourcing-based	[19], [20], [21], [22], etc	human judgment	account	with considerable delay	No
Traffic-flow-based	**BotFlowMon**	*network flows*	*behavior*	*real time*	*Yes*

flows, and connections. Later, more approaches, such as those in [42], [43], [44], used increasingly popular NetFlow data to classify traffic. Compared with BotFlowMon which aims to classify traffic from the same application (traffic from social bots vs. traffic from human users), none of these approaches classify traffic from different groups of entities within the same application.

Like BotFlowMon, many traffic anomaly detection approaches are content-agnostic and use network flow information as input, such as those that detect distributed denial-of-service (DDoS) [45], botnet [46], worm [47], and cryptojacking [48]. However, the social bot anomaly that BotFlowMon tries to detect differs from these anomalies in fundamental ways, warranting a completely different detection approach. Foremost, any social bot anomaly is specific to an OSN application; to detect it using network flow records, one must extract OSN-specific behaviors from the records and detect behaviors that are anomalous . Existing content-agnostic traffic anomaly detection approaches do not study OSN-specific anomalies, making them unsuitable for detecting social bot traffic. Moreover, traffic caused by anomalies such as DDoS, botnet, worm, or crypto-mining is usually of different protocols, destinations, and underlying applications than those of legitimate traffic; social bot traffic, however, usually use the same protocols (e.g., HTTPS), destinations (e.g., Facebook), and underlying applications (OSN) as those of real OSN user traffic, making these attributes unusable for social bot detection. In fact, every content-agnostic traffic anomaly detection approach is designed for a specific type of anomaly and hardly interchangeable. For example, whereas content-agnostic botnet detection could use the trace of command and control channels (e.g., [49], [50]), IRC messages (e.g., [51]), or collective DNS queries (e.g., [52]), none of these data or their properties applies to the detection of social bots.

3 BotFlowMon Design

3.1 Overview

In order to detect if any machine in a network is a social bot and producing social bot traffic, BotFlowMon can be deployed on any machine that can access and analyze the traffic flow data between the monitored network and OSN servers. Figure 1 shows two different operational models of BotFlowMon, one with BotFlowMon accessing traffic from the router of the monitored network at the source end (e.g., a campus network), the other from the router of an OSN server at the destination end (e.g., Facebook).

There are different traffic flow formats. We focus on the NetFlow [8] format, which is widely used for network traffic monitoring and analysis. Our design can easily extend to other flow formats such as sFlow [53].

Every NetFlow record logs information of a network flow inbound or outbound, including its start time, end

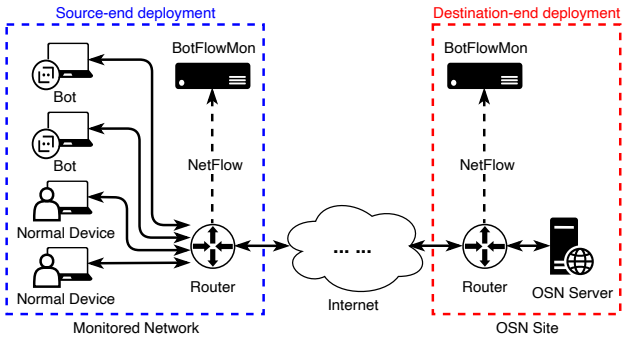


Fig. 1. Operational models of BotFlowMon.

time, number of packets, and number of bytes. It also records information from the TCP/UDP and IP headers of all the packets in the flow, including source IP address, destination IP address, protocol, port numbers, type of service (ToS), and TCP flags. It will not access, read, or record the payload of any packet where the OSN content would be carried. As a result, a NetFlow record will *not* contain OSN content data such as OSN account profiles, Facebook or Twitter messages, or posted images. In another words, NetFlow records are content-agnostic. Using NetFlow records as input is thus privacy-preserving. Figure 2 shows the fields of NetFlow records that BotFlowMon uses, with a few examples.

Figure 3 illustrates the architecture of BotFlowMon. It encompasses two modes: **training mode** which uses labeled NetFlow data to derive a classification model and **detection mode** which uses the classification model to detect social bot flows from the input traffic flows. It also consists of *five* modules: preprocessing, flow aggregation, transaction fingerprint generation, transaction subdivision, and machine learning & classification. We detail each module below.

3.2 Preprocessing

Motivation. With the raw NetFlow records collected from a router as input, the preprocessing module needs to select records only related to OSN, group them by OSN users, and output them to the next module. As a router forwards packets and summarizes them into NetFlow records, there can be a vast amount of NetFlow records generated every second and these records can be noisy as they contain flows not toward OSNs or flows of irrelevant protocols and applications. The preprocessing module therefore must be both efficient and accurate in extracting OSN-related flows.

Design. We preprocess the raw NetFlow data collected from a router as follows. We first extract the traffic flows only related to OSNs. After removing NetFlow records with zero bytes, zero duration, or irrelevant protocols (such as ICMP), BotFlowMon further discards NetFlow records

Start Time	End Time	Protocol	Src IP	Dst IP	Src Port	Dst Port	TCP Flags	ToS	pkts	bytes
1582822775.808	1582822802.688	TCP	240.246.127.7	43.175.217.143	80	17608	...A....	56	1300	1846000
1582822776.832	1582822782.208	UDP	28.7.12.109	2.50.210.91	443	49599	....	8	8192	10485760
1582822421.713	1582822425.191	TCP	13.187.25.153	123.41.10.88	443	34530	...A....	80	120	37920

Fig. 2. Fields of NetFlow records used by BotFlowMon.

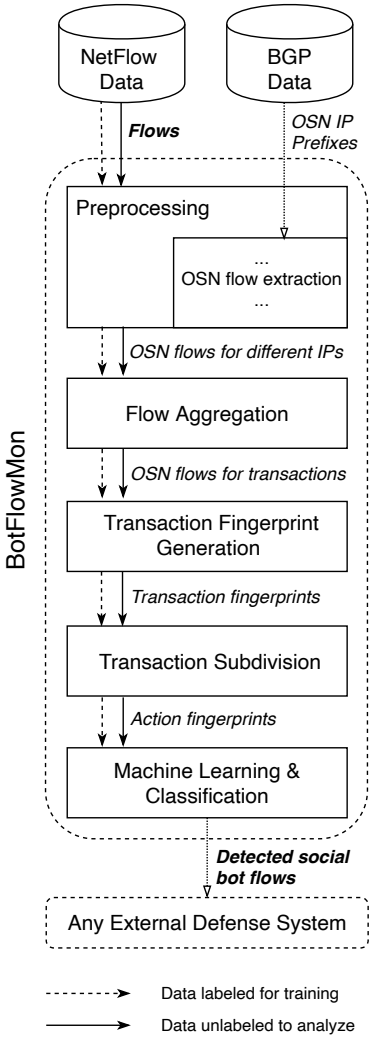


Fig. 3. BotFlowMon architecture. It has two modes of operation: training (data in dashed line) and monitoring (data in solid line).

whose source or destination IP address is not associated with the social network site(s) in question (e.g., Facebook or Twitter). One issue here is that each OSN site may own hundreds or even more IP prefixes and they may change over time. We leverage BGP stream [54] to obtain an OSN site's current IP prefixes and check if a NetFlow record's source or destination IP address matches one of the prefixes. In order to deal with a large number of flows efficiently, the matching process utilizes the longest prefix match algorithm [55], which is similar to the IP forwarding table lookup procedure when a router forwards packets based on their destination IP address. Our design allows BotFlowMon to preprocess NetFlow records in real time. We then group NetFlow records by OSN users, since these records may summarize traffic from thousands of users. Here, we define each OSN user by a unique combination of an IP address and a port number from the monitored network.

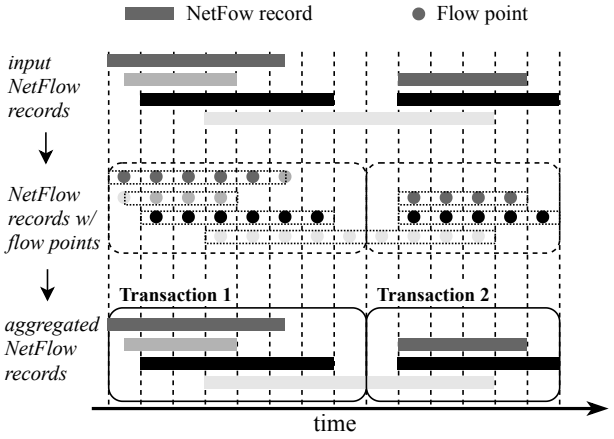


Fig. 4. A flow aggregation example using modified DBSCAN. Every time bin is a small window (e.g., 0.1 seconds). Six NetFlow records are clustered into two transactions by converting them to NetFlow records with flow points. (Each flow point has a different gray level, with a darker gray indicating a higher traffic volume.) One NetFlow record was fragmented into two records, each at a different transaction.

3.3 Flow Aggregation

Motivation. Now that we have preprocessed NetFlow records to be composed of only those relevant to detecting social bot flows, we address the next challenge in that there is no sufficient information from data of *individual* NetFlow records to distinguish social bot flows from OSN flows generated by real users. As both social bot and real OSN user behaviors are conducted at the application level, their NetFlow records, which do not record application-specific data, can easily be indistinguishable.

Design. We thus introduce the flow aggregation module in BotFlowMon to inspect *collective* OSN behaviors of flows in order to capture distinct patterns of social bot versus real user behaviors. It aggregates all the NetFlow records generated by the same **transactions**, so that we can inspect and compare transactions of a social bot against those of real users, including defining and comparing features of transactions. Here, a transaction is a sequence of actions by either a social bot or real user that are closely adjacent to each other. For example, it can be a user logging in her Facebook account and reading new posts on her Facebook wall, or a Twitter bot retweeting a spam link one hundred times within a short period.

In this module, we use modified DBSCAN [56] to aggregate/cluster flows into multiple transactions, with each transaction composed of multiple flows. Figure 4 shows an example. DBSCAN is a common density-based clustering algorithm that groups together adjacent high-density data points, where outliers are points that lie only in low-density regions. This procedure includes the following steps:

1) For every flow as described by a NetFlow record, we divide its duration into multiple time bins of equal length and define a flow point for each bin. Every flow point has a traffic volume derived from the bits per

TABLE 2
$6 \times N$ matrix as a transaction fingerprint.

Features	Values			
1: outgoing bps	bps^o_{t1}	bps^o_{t2}	...	$bps^o_{t_N}$
2: outgoing pps	pps^o_{t1}	pps^o_{t2}	...	$pps^o_{t_N}$
3: outgoing ToS	tos^o_{t1}	tos^o_{t2}	...	$tos^o_{t_N}$
4: incoming bps	bps^i_{t1}	bps^i_{t2}	...	$bps^i_{t_N}$
5: incoming pps	pps^i_{t1}	pps^i_{t2}	...	$pps^i_{t_N}$
6: incoming ToS	tos^i_{t1}	tos^i_{t2}	...	$tos^i_{t_N}$

second (bps) of the flow multiplied by the bin's length.

2) We run the modified DBSCAN algorithm to group flow points into clusters, with each cluster composed of flow points that are closely adjacent to each other over a time window and have a high total traffic volume. In particular, for any time interval of ϵ seconds from the time window, the flow points that belong to the interval have a total traffic volume no less than $minPts$ bits.
3) Finally, we inspect the time window of every cluster from the above step. All the flows that fall within this window will belong to the same transaction.

3.4 Transaction Fingerprint Generation

Motivation. With multiple transactions obtained from the flow aggregation module, every transaction may contain a different number of NetFlow records, further with its information in textual format (as every NetFlow record is textual). To make different transactions directly comparable to each other, BotFlowMon must define, extract, and normalize features from the aggregated flows of each transaction.

Design. BotFlowMon generates a **fingerprint** for each transaction. We design a **data fusion method** that derives an $f \times N$ matrix from every transaction and use this matrix as the fingerprint of the transaction. Here, N is the number of time bins of equal length within the time window of the transaction, which spans from the earliest start time to the latest end time among all flows in the transaction, and f is the number of features of the transaction over each time bin.

Table 2 shows a $6 \times N$ example transaction fingerprint matrix. Rows 1 to 3 are features of outgoing flows and rows 4 to 6 are features of incoming flows. Both use bits per second (bps), packets per second (pps) and type of service (ToS) as features. Note for any time bin there can be more than one flow active, thus the values of bps and pps (either incoming or outgoing) for that time bin should be respectively the sum of the bps and pps values of all flows active in that time bin. The outgoing or incoming ToS feature for a time bin, however, is not numerical, and its value is the ToS field (which is widely used for prioritizing traffic) and TCP flag of the flow that has the largest bps value during the time bin. However, because the usage of the ToS field has not been standardized, the ToS feature may not be reliable to help produce a transaction fingerprint. As such, we also introduce a $4 \times N$ matrix that does not include the ToS feature for incoming and outgoing flows.

Once a transaction fingerprint matrix is generated, it also must be normalized. In BotFlowMon, we use the quantile normalization approach to map *every* value in the matrix to a number between 0 and 255. Using the $6 \times N$ matrix from Table 2 as an example, we learned the distributions of bps, pps and ToS values using a 235-GB NetFlow dataset of campus traffic, obtained three functions $f_r(bps)$, $f_g(pps)$, and $f_b(tos)$ based on the distributions, and then normalized the bps, pps, and ToS values in the matrix to numbers between 0 and 255, respectively.

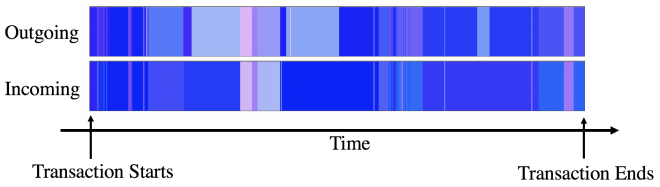


Fig. 5. Visualizing the fingerprint of a transaction lasting 35.74 seconds with 220 flows.

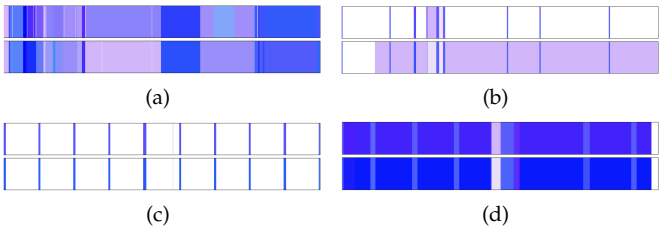


Fig. 6. Transaction fingerprint image examples.

With all the values in a $f \times N$ transaction matrix between 0 and 255, we can easily visualize it for inspection and analysis. We can generate an image composed of two colorful bars in the standard RGB space, one for the incoming flows in the transaction and one for the outgoing flows, where each bar has a length of N pixels. In particular, we calculate the RGB value of pixel i (i=1,...,N) using all the values from the i-th column of the matrix. For example, for the $6 \times N$ matrix in Table 2, we can use the outgoing bps, pps, and ToS values in column i to derive the i-th pixel for the outgoing bar in the image as (R, G, B) = (255-bps, 255-pps, 255-ToS), which maps a bigger bps or pps value to a darker color.

Figure 5 is an example of visualizing a transaction fingerprint, which has 220 flows and represents a real user spending 35.74 seconds browsing Facebook. The beginning and ending positions of the image correspond to the starting and ending time of the transaction, respectively, with the upper bar about the outgoing traffic and the lower bar about the incoming traffic.

Figure 6 shows more examples of visualized transaction fingerprints from labeled ground truth. Transaction fingerprints in Figures 6(a) and 6(b) are two Twitter users messaging each other through Twitter Direct Messages, where Figure 6(a) is generated by a real user and Figure 6(b) is generated by a chatbot. Figure 6(c) is created by a social bot that uses APIs to tweet text messages every 3 seconds. Figure 6(d) is created by a bot that crawls the photo albums of friends and posts spam links at the same time. We can clearly see distinguishable patterns between the transaction fingerprints of a social bot and those of real users in terms of color depth, frequency, regularity, and density.

3.5 Transaction Subdivision

Motivation. Now that we have produced a normalized fingerprint matrix for every transaction, we observe there can

be countless types of transactions since every transaction may contain an arbitrary number of actions of various types. In addition, each transaction can be of an arbitrary duration ranging from a few seconds to a few hours, which can lead to a huge amount of training data if to have enough training data for every possible duration range.

Design. We thus subdivide a transaction further into a sequence of primitive, short-lived behaviors called **actions**. Compared to countless types of transactions, there are only a limited number of types of actions, such as clicking a Like button, sending a tweet, or submitting a comment. As we will illustrate, as opposed to inspecting their transactions, it is much easier to differentiate social bots from real users through their actions in transactions. Once we tell social bot actions apart from real user actions, we can separate social bot transactions from real user transactions.

To subdivide a transaction into actions, we design a new clustering algorithm named **density-valley-based clustering** for this purpose. Compared with other density-based clustering algorithms such as DBSCAN [56] and OPTICS [57] that work by traversing density-connected areas, our algorithm clusters data points by finding the density valley between adjacent clusters. It does not require a density threshold parameter to conduct the clustering. Instead, it uses a valley point index ρ to identify the boundary of two clusters. Besides, our algorithm has a good performance in processing datasets whose density is not uniform.

This algorithm uses the following terms:

1) **Density of a data point:** The density of a data point p is the summation of the values of all data points within a radius of r from p. Using bps values as an example, $p.density = \sum_{dist(x,p)<r} x.bps$.
2) **Density of a cluster:** A cluster's density is the density of the point in the cluster that has the highest density.
3) **Potential point of a cluster:** Point p is a potential point of C if p is within radius r of a point $b \in C$.
4) **Valley point:** A data point p is a valley point of multiple clusters if p is a potential point of each of these clusters.
5) **Valley point competition:** When two clusters share a valley point, they "compete" to include the valley point as its member, with two possible outcome:
   a) The two clusters merge into a new cluster, with the valley point now belonging to the new cluster;
   b) The two clusters keep separate, with the valley point assigned to the cluster with less data points.

   Here, the two clusters keep separate if the density of the valley point is lower than a percentage ρ of, thus in sharp contrast to, the density of both clusters.

The algorithm works as follows, with its pseudocode in Algorithm 1. It takes a dataset D and a **radius threshold value** (r) as input. Using the $6 \times N$ or $4 \times N$ matrix from Section 3.4 as an example, D can be a set of data points where the i-th data point has a bps value that is the sum of the outgoing bps and incoming bps from the i-th column of the matrix. The algorithm then sorts all the data points in D, processes all the data points in the descending order of density, and forms and populates clusters with data points in D. If a data point is a valley point between two clusters, the algorithm then decides whether to merge the two clusters or still keep them using the valley point competition mechanism; if the two clusters do not merge, we also identify the subdivision moment where the two clusters meet. Finally, the algorithm outputs all the newly formed clusters. If D is a set of data points from a transaction's fingerprint, these clusters then represent actions of the transaction.

Algorithm 1 Density-valley-based clustering algorithm.

Input: dataset D, radius threshold value r, valley point index ρ
$C = \phi$ ▷ C is a set to store clusters
Use r to calculate the density of each data point in D
$D := Sort(D)$ ▷ Sort data in the decreasing order of density
for data point e in D **do**
  if e is not a potential point of any cluster **then**
    Label e as a member of a new cluster c_e
    Add cluster c_e to C
  else if e is a potential point of cluster c_a **then**
    Label e as a member of cluster c_a
  else if e is a potential point of two clusters $\{c_i, c_j\}$ **then** ▷ Start the valley point competition
    if $e.density \leq \rho \cdot min(c_i.density, c_j.density)$ **then**
      Add e to c_i or c_j that has less data points
    else
      $c_{new} = merge(c_i, c_j)$
      Add data point e to cluster c_{new}
      Remove clusters c_i and c_j from C
      Add cluster c_{new} to C
    end if
  end if
end for
return C

After subdividing a transaction into actions using the algorithm, we further ensure every action is short-lived according to the definition above. If an action produced from the algorithm is longer than 60 seconds, we further divide it into multiple actions at 60-second intervals.

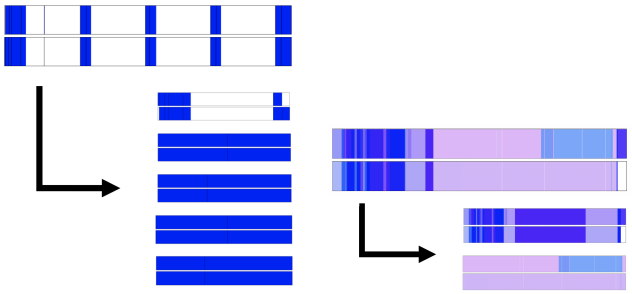
Like a transaction represented by a transaction fingerprint matrix, an action can be represented by an **action fingerprint matrix**, which has the same look as a transaction fingerprint matrix as shown in Table 2.

Figure 7 shows two transaction subdivision examples. Figure 7(a) shows by subdividing the transaction fingerprint of a social bot (which is a post bot) into five actions, every action has a more outstanding pattern than the original transaction fingerprint. Figure 7(b) shows a transaction by a real user that is composed of two actions, where one was opening an OSN site and the other was scrolling down the page of the OSN site. We can see actions from real users present more complicated fingerprint images than those from social bots, making them easy to differentiate.

Algorithm 1 is dedicated to low dimensional datasets, such as the transaction fingerprint data. We further extend it in Appendix A to make it a general clustering algorithm that can deal with multi-dimensional datasets.

3.6 Machine Learning & Classification

Motivation. With transactions subdivided into actions, the main challenge now becomes classifying transactions



(a) Social bot (b) Real user

Fig. 7. Transaction subdivision examples.

through their actions, while it is yet to be seen what models work best for classifying actions. Also, we need to devise an architecture when applying a model to process action fingerprints.

Design. We first classify actions into social bot actions and real user actions, and then classify the transactions based on how their actions are classified.

Action classification model. To classify actions we train an action classification model. The input to this model is a set of action fingerprint matrixes labeled as either social bot actions or real user actions. Since every action fingerprint matrix is nonlinear and high dimensional, we use Conventional Neural Network (CNN) [58] and Multilayer Perceptron (MLP) [59] as classification models. (BotFlowMon is not bound to MLP and CNN and could use other machine learning algorithms if applicable.)

CNN and MLP architecture. We use the Keras [60] library with TensorFlow [61] to implement CNN and MLP architectures. Figure 8 shows a CNN architecture that is composed of a convolution layer, a pooling layer and a flatten step to transform a fingerprint matrix into the input of a fully connected neural network, which consists of an input layer, multiple hidden layers (we evaluate the optimal number of hidden layers in Section 4.2.3), and an output layer to finally output the label for the fingerprint matrix. The MLP architecture is similar to the fully connected neural network in the CNN architecture, with the input layer being a flattened fingerprint matrix. For both architectures, the hidden layers use Leaky ReLU as its activation function so the model can converge quickly; the output layer uses the Sigmoid function to produce a probability that the fingerprint matrix is from a social bot. In addition, we train the CNN and MLP models with the stochastic gradient descent optimization algorithm and use the backpropagation algorithm to update the neural network.

Transaction Classification. With the action classification model trained, we then can classify if every action within a transaction is from a social bot or from a real user and decide whether the transaction is a bot transaction or a real user transaction. Apparently, if all actions are from a real user (or a social bot), we can safely determine the transaction is by a real user (or a social bot). However, a transaction may consist of actions of both types. We thus decide that a transaction is a bot transaction if more than a certain percentage of actions are from a social bot, and a real user transaction if otherwise. We define this percentage as the detection sensitivity rate γ of BotFlowMon.

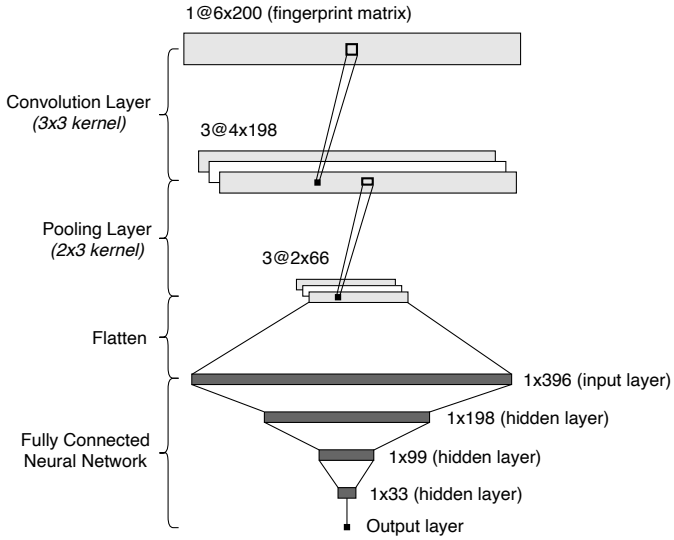


Fig. 8. The CNN architecture of BotFlowMon (with example parameters).

4 Evaluation

We now evaluate BotFlowMon. We first introduce the dataset used in Section 4.1, then config and analyze parameters used in the BotFlowMon system in Section 4.2, present detection results and analysis in Sections 4.3–4.6, and finally discuss BotFlowMon's performance in Sections 4.7.

4.1 Data Source

The datasets we use to construct and test BotFlowMon come from two sources: (i) the traffic generated and gathered from our lab's computers and routers, which is a small experimental platform that has superior flexibility and convenience for simulation, data collection, and experiments; and (ii) datasets generated and collected from the campus network traffic of a large university, which offers data from realistic scenarios for analysis and verification.

We created and labeled the real user and social bot traffic flows as follows. For real user traffic flows, we recruited participants to manually conduct normal daily activities on Twitter and Facebook using our lab's computers. For social bot traffic flows, we deployed open-source social bot programs and homegrown bot scripts on the experimental platform and the campus network to conduct bot activities on Twitter and Facebook. We then collected and labeled the corresponding traffic as the ground truth. As described in Appendix B, we categorized social bots into five types according to their implementation mechanisms and simulated all of them. Besides, since the development and evaluation of BotFlowMon involved human subjects, we address the ethical and human subject issues of this process in Appendix C.

Table 3 shows the composition of the collected data. We collected 28 GB raw NetFlow data from our experimental platform and 507 GB raw NetFlow data from the campus traffic, both containing all traffic flows in the environment. After preprocessing, we had a dataset of 3.204 GB with

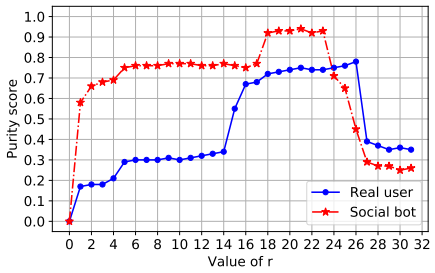


Fig. 9. Purity scores with different r values.

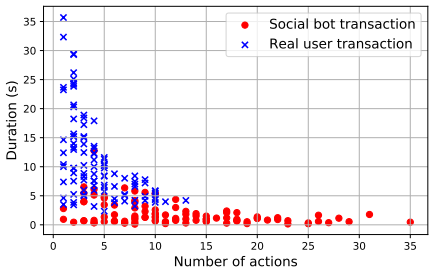


Fig. 10. Scatter diagram for subdivision.

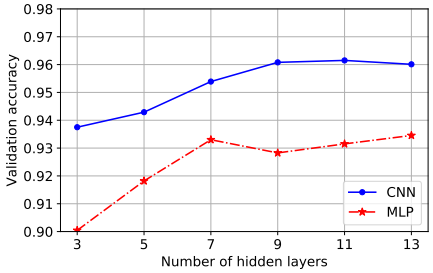


Fig. 11. Validation accuracies with different numbers of hidden layers.

TABLE 3
Composition of collected data.

Raw NetFlow records			
Size of data	535 GB (campus network: 507 GB, lab platform: 28 GB)		
Labeled NetFlow records			
Size of data	3.204 GB (987.710 MB if removing irrelevant fields)		
# records	30,932,991		
Number of transactions (social bot : real user ≈ 7:3)			
Social bot	166,615	Real user	67,723

30,932,991 labeled NetFlow records for training and testing. The ratio of social-bot transactions to real-user transactions is approximately 7:3.

4.2 System Parameters Configuration and Analysis

The BotFlowMon system includes multiple system parameters for its different modules. We first set up the empirical or default values for some system parameters in Section 4.2.1. We then investigate two key parameters more specifically: the radius threshold r in the transaction subdivision module in Section 4.2.2 and the number of hidden layers in CNN and MLP in the machine learning & classification module in Section 4.2.3.

4.2.1 System Parameters with Empirical or Default Values

Based on our empirical studies (of which we skip the details for space considerations), we set the following parameters as follows:

- For the flow aggregation module, we set the length of every time bin to be 0.1 seconds. This length provides a time granularity that is fine enough but not too small to skyrocket the computation cost of the system. This parameter is adjustable, as smaller time bins could lead to more precise results with a higher computational cost.
- For the DBSCAN algorithm in the flow aggregation module, we set ϵ to be 10-20 seconds. According to the user behavior models from the network perspective [62], users usually do not trigger NetFlow records in 10-20 seconds. This setup also helps us lean toward clustering the flows into transactions over a longer period rather than short ones, as the former is more friendly with the transaction subdivision module. Further, we set $minPts$ to be 1500 bits, which is a relatively small value for OSN traffic [62] in order to accommodate certain tiny streams between an OSN user and an OSN server.
- For the transaction fingerprint generation module, we set N as 200. Note a larger value of N will generate more accurate results but require more training data and computations.

We also set the default values of the following parameters:

- For the transaction subdivision module, we define ρ in valley competition mechanism to be 50% by default.
- For the machine learning & classification module, we set the detection sensitivity rate γ at 50% by default.

4.2.2 Subdivision Efficacy and Its Radius Threshold (r)

We evaluated the transaction subdivision module to see whether our density-valley-based clustering algorithm can divide the transaction fingerprints into action fingerprints correctly. Specifically, we studied how different values of radius threshold r in the algorithm could generate different subdivision results and potentially affect the detection outcome. Figure 9 shows the purity scores of the resulted clusters from subdivision with different r values. We use this formula to calculate the purity scores: $purity = \frac{1}{N}\sum_k max_j |c^k_{truth} \cap c^j_{algo}|$, where N is the number of input data points, c^k_{truth} is the k-th cluster from the ground truth, and c^j_{algo} is the j-th cluster generated by the algorithm. From Figure 9, we can see that the algorithm is indeed susceptible to the values of r and we can generate optimal results when r is in the range of 18 to 23. For transactions of social bots, the subdivision module works well when r is in this range and can achieve more than 0.90 purity of the resulting clusters. This is because social bots utilize APIs heavily, making their transaction fingerprints easy to subdivide. For real user transactions, however, the purity scores of the clusters derived from the algorithm are lower and more sensitive to the variation of r than those corresponding to bot transactions, with the maximal purity score to be 0.7832 when $r = 26$. One reason is that the boundaries of different actions in real user transactions are blurry in flow-level data. Almost all the OSN web sites preload content to real users dynamically, which can cause irrelevant NetFlow records to occupy the gap between actions and thus make the clustering in the algorithm less accurate.

However, the ultimate goal of subdivision is not partitioning all the transactions precisely. Instead, it is designed

to make data more friendly to the machine learning process. We randomly sampled 100 real user transactions and 100 bot transactions, then recorded the number of actions and their average duration for each transaction after subdivision. Figure 10 is the scatter diagram of the result. We found that while the lengths of actions vary, actions from bots tend to have a short duration (0 to 15 seconds) and actions generated by real users usually have a long duration (0 to 40 seconds). Nonetheless, the durations of bot actions and real user actions are *comparable*, which makes it easy to compress actions of different durations into fingerprints of a fixed length *and* makes the subsequent machine learning module easy to converge.

4.2.3 Number of Hidden Layers in CNN and MLP

We investigated the optimal numbers of hidden layers of our CNN and MLP models as this parameter can significantly affect the accuracy of the models. In this evaluation, we took 80% of the labeled data as the training dataset, 10% of the data as the validation dataset, and the remaining 10% of the data as the testing dataset. For both CNN and MLP models, we started with three hidden layers. We trained each model with the training dataset and validated their accuracies with the validation dataset. We then kept adding new hidden layers to each model and repeated the procedure above until the validation accuracies converge.

Figure 11 shows the validation accuracies with different numbers of hidden layers. We can see the validation accuracy of CNN is already acceptable when the number of hidden layers is three and remains almost the same after it reaches nine. The MLP model's accuracy stabilizes when the number of hidden layers reaches seven. Therefore, we set CNN's default number of hidden layers at *nine* and MLP's default number of hidden layers at *seven*.

4.3 Classification Accuracy

We evaluated different classification models' abilities in detecting social bot transactions. Figure 12 shows the test results for machine learning & classification module with different classification models. The test dataset contains 10% of the labeled data, which has 16,661 social bot transactions and 6,772 real user transactions. Here, we use 6×200 transaction fingerprints which use incoming and outgoing bps, pps and ToS features. Both CNN and MLP models can correctly classify more than 93% of the traffic flows. However, CNN achieves a better significant result, with 96.1% of accuracy. One reason for this phenomenon is that CNN has convolutional and pooling layers to incorporate spatial information from action fingerprints, while MLP only takes flattened vectors as input, which disregards such spatial information. Moreover, we can clearly see that the subdivision procedure helps improve the accuracy for more than 10%. Table 4 shows other detailed evaluation scores of MLP and CNN for both 6×200 and 4×200 transaction fingerprints. We can see that CNN slightly surpasses MLP under all scoring criteria.

Overall, BotFlowMon can detect social bots with high accuracy. With low false negative (or high true positive) rates as shown in Table 4, we can detect every single social bot transaction with a high probability. In particular, because a social bot usually creates multiple transaction fingerprints in a burst and we often only need to identify one of them to be able to trace the social bot, our chance of detecting the social bot is even higher.

TABLE 4
True/false positive/negative rates for social bot traffic detection.

	MLP		CNN	
	6×200	4×200	6×200	4×200
True positive rate	0.9358	0.9318	0.9665	0.9485
True negative rate	0.9260	0.9221	0.9468	0.9334
False positive rate	0.0739	0.0778	0.0531	0.0665
False negative rate	0.0641	0.0681	0.0334	0.0514

A concern here is the false positive rates as we do not want BotFlowMon to mislabel real user traffic. An active social network user may generate hundreds of transactions per day, which means the false positive rate of the detection system needs to be low enough to avoid causing interruptions to real users. For this purpose, we can leverage the detection mechanism in the machine learning & classification module (Section 3.6); in particular, we can adjust the detection sensitivity rate γ to achieve zero false positive rate, which we further investigate in Section 4.4.1.

4.4 Putting BotFlowMon to Use

We now study if putting BotFlowMon to use, what factors the user can config and adjust to suit their environment and need and how they may affect the detection of social bots.

4.4.1 Detection Sensitivity Rate (γ)

BotFlowMon allows its detection sensitivity rate γ to be adjusted to achieve different false positive rates. As stated in Section 3.6, given a sensitivity rate γ, BotFlowMon identifies a transaction as a social bot transaction if the percentage of social bot actions in the transaction is larger than γ. If we want to limit or eliminate false alarms, we can increase γ to sacrifice the detection accuracy a little bit for an extremely low (or even zero) false positive rate.

Figure 13 shows the accuracies and false positive rates of BotFlowMon with both MLP and CNN models when different γ values are used. We found from our experiments that the false positive rate of CNN will reach nearly zero when γ is no less than 0.75 (while the true positive rate is 0.91). Similarly, the false positive rate of MLP will reach zero when γ is no less than 0.85. If only when more than 75% of its actions are classified as bot actions will a transaction be labeled as a bot transaction, BotFlowMon with CNN will not generate any false alarm, while its accuracy will only drop down to 0.89. Therefore, a conservative approach to deploying BotFlowMon is to set the detection sensitivity rate γ at 0.75 if using CNN (or 0.85 if using MLP), in which case the detection accuracy will be more than 0.89 if using CNN (or 0.85 with MLP) with no false positives.

4.4.2 ToS Features

Due to reliability concerns with the ToS features (see Section 3.4), BotFlowMon may need to run without such features. We investigated BotFlowMon's detection efficacy without them. We thus used 4×200 transaction fingerprints

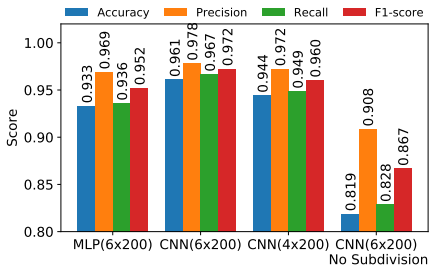


Fig. 12. Detection scores for different detection models.

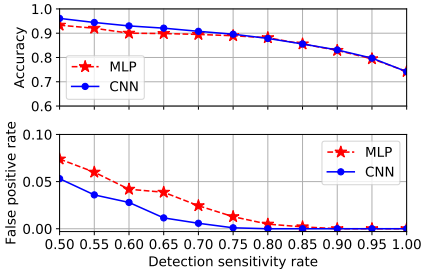


Fig. 13. Detection accuracy and false positive rate with different detection sensitivity rates (γ).

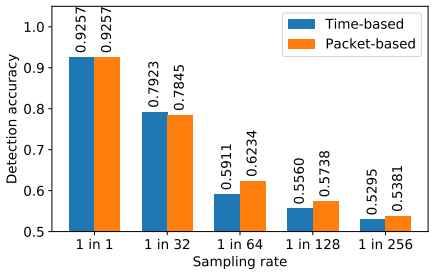


Fig. 14. Detection accuracy with sampled NetFlow records (CNN with three hidden layers).

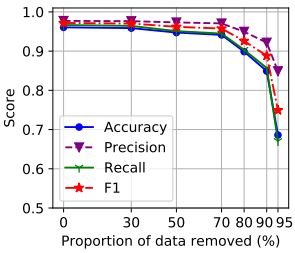


Fig. 15. Detection scores with reduced training data.

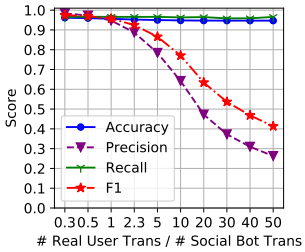


Fig. 16. Detection scores with imbalanced test datasets.

which only included incoming and outgoing bps and pps features. Figure 12 shows the results, where the overall accuracy is 0.944, which is 1.7% lower than that with 6×200 transaction fingerprints. We can see without ToS features, we can have a classification model that is not only more usable but also still fairly accurate.

4.4.3 Detection with Sampled NetFlow Records

A reality factor in running BotFlowMon against NetFlow records as input is that many times due to computational and bandwidth pressure on routers, only sampled NetFlow records are available. We thus investigated BotFlowMon's accuracy with sampled NetFlow records.

We applied two sampling methods over the original NetFlow data from Table 3: time-based sampling that takes packets from a τ-second interval every $\tau \cdot x$ seconds, and packet-based sampling that takes one packet every x packets. We created multiple datasets using both methods with different sampling rates (i.e., $\frac{1}{x}$) and run BotFlowMon using the CNN classification model with three hidden layers. Figure 14 shows the results. We can see that as the sampling rate decreases the accuracies drop dramatically. BotFlowMon still works well with a low sampling rate at 1 in 32, but it becomes barely usable when the sampling rate becomes 1 in 64 or lower. On the other hand, as we pointed out in Section 4.3, because a social bot usually spawns multiple transaction fingerprints in a burst, among which only one needs to be identified to track down the social bot, we still have a reasonably high chance to include at least some bot transactions in the sampled input and be able to detect at least one of them.

4.4.4 Detection with Less Training Data

It is laborious to train a classification model with a large amount of labeled data. We investigated BotFlowMon's capability with less training data to understand the least amount of training data that would still lead to acceptable performance. Specifically, we removed different portions of data from the original training dataset and evaluated BotFlowMon's detection ability under each different training data size with default parameter values. Figure 15 shows the results. We can see BotFlowMon maintains a decent accuracy even after we removed 90% of the training data, but its accuracy degrades significantly when it is more than 90%. In another words, to maintain satisfactory performance the training needs at least 1500 transactions (250 MB) from real users and also this much from every type of social bot.

4.4.5 Detection with Imbalanced Datasets

When deploying BotFlowMon in different environments, the ratio of real user transactions to social bot transactions can change significantly. For instance, the proportion of social bot transactions in a VPS provider's network can be higher than that in residential networks. We therefore generated ten test datasets with different ratios to evaluate BotFlowMon's detection performance under such scenarios with default parameter values. Each of the generated test sets has 3000 samples. Figure 16 describes BotFlowMon's accuracy, precision, recall, and F1 scores with these test datasets. We can see that when the proportion of real user transactions increases, the accuracy and recall remain almost the same, but the precision and F1 score clearly drop. This phenomenon is expected as the false positive samples will dominate the detection errors when the number of false samples grows. To solve this problem, we can adjust the detection sensitivity rate discussed in Subsection 4.4.1 to reduce false alerts.

4.4.6 Selection of Parameters in New Environments

When transferring BotFlowMon to a different network environment, users do not need to reset the parameters if the following two conditions are met: (1) The input NetFlow records are of the same version and configuration. (2) Users are going to detect the same types of social bots on the same OSN sites. Otherwise, users need to reset the parameter values or retrain the detection model. To quickly set up the parameters in BotFlowMon and achieve acceptable performance in such scenarios, we suggest the users to begin with default parameters and use a conservative detection sensitivity rate γ (e.g., 0.85 - 0.95). Such setting will sacrifice the true positive rate to decrease the false positive rate, enabling

users to detect social bots to some extent while not being overwhelmed by false alarms. Later, after other parameters (e.g., the neural network) are fine-tuned according to the environment, users can adopt a radical detection sensitivity rate (e.g., 0.5 - 0.75) for better accuracies.

4.5 Comparisons with Other Approaches

We have compared BotFlowMon with other social bot detection approaches, specifically content-based methods and structure-based methods (elaborated in Section 2). As illustrated in Figure 17, they each operate using different input at a different granularity in order to detect social bots.

We compared BotFlowMon with a content-based method described in [63], which we refer as SpamFilter. It utilizes four types of content features to detect social bots and achieves a true positive rate (TPR) of 80.8% and a false positive rate (FPR) of 0.32%. By analyzing their experimental data, an OSN account has to generate at least two transactions in order to create enough data to derive values of these features for SpamFilter to conduct its detection. We thus compare BotFlowMon with SpamFilter with two transactions.

As for structure-based methods, we selected SybilWalk [17] to compare, which achieves a TPR of 96% and a FPR of 1.3%. Here, in order to have enough data to derive values of features used by SybilWalk, an OSN account has to conduct at least four transactions, including account creation, profile initiation, and relationship establishment. We thus compare BotFlowMon with SybilWalk with four transactions.

Figures 18(a) and 18(b) illustrate the comparison results, where BotFlowMon uses the CNN model with 6×200 fingerprints and sets the detection sensitivity rate γ at 0.75. We can see the following. When there is only one transaction, only BotFlowMon can detect whether it is a social bot transaction with a TPR of 91% and a FPR of 0.1%. When there are two transactions, BotFlowMon can detect whether there is at least one social bot transaction with a TPR of 99.1% and a FPR of 0.2%, which is significantly better than SpamFilter's TPR (80.8%) and FPR (1.3%). And when there are four transactions, BotFlowMon can detect whether there is at least one social bot transaction with a TPR of 99.9% and a FPR of 0.5%, which is still significantly better than SpamFilter's TPR (96.3%) and FPR (2.6%) and is also better than SybilWalk's TPR (96.0%) and comparable to its FPR (0.3%). Overall, BotFlowMon outperforms both SpamFilter and SybilWalk with less transactions needed, a higher TPR, and a higher or comparable FPR.

4.6 Feasibility Study on Multiclass Classification of Social Bot Traffic

Besides identifying social bot traffic flows, we investigated whether BotFlowMon may further classify the social bot traffic flows into different categories of social bots. By using the transaction fingerprints of different categories of social bots, we trained a multiclass CNN classification model that labels a social bot transaction fingerprint as one of the five following categories: chatbot, post bot, amplification bot, OSN crawler, and hybrid bot.

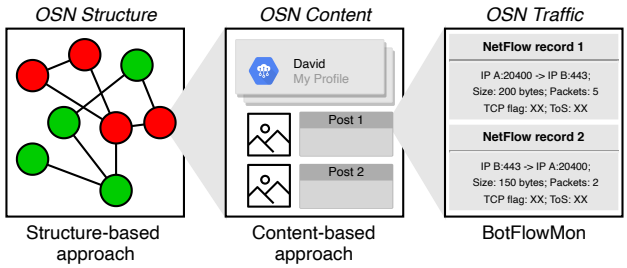
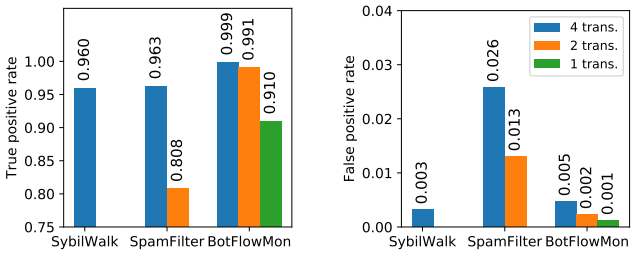


Fig. 17. Input comparison of different social bot detection approaches.



(a) True positive rates (b) False positive rates

Fig. 18. BotFlowMon vs. other approaches.

Figure 19 shows the multiclass classification scores for different categories of social bots. We used a test dataset that contains 250 bot transactions, with 50 transactions for each category. The overall classification accuracy is 78%. We can see from Figure 19 that this model achieves the best score for labeling OSN crawlers. This is because OSN crawlers usually have high frequencies, large packet sizes, and sharp contrast of outgoing and incoming traffic volumes, leading to the most differentiable traffic flows among the five types of social bots. The scores for chatbots and hybrid bots are less remarkable, but still decent. On the contrary, it is relatively hard to distinguish traffic flows from post bots and amplification bots; the API calls made by both types of bots, regardless of the functions invoked (e.g., liking a post, posting a message, following an account), will result in almost identical traffic flows.

In general, the accuracy for classifying categories of social bots is significantly lower than that of detecting social bots. These two tasks are fundamentally different. The former aims to classify the traffic as different types of social bot traffic, while the latter is focused on classifying the traffic as real-user traffic or social-bot traffic. Obviously, the difference between real-user traffic and social-bot traffic is more significant and easier to distinguish.

4.7 Time Complexity and Performance

4.7.1 *Time Complexity*

The time complexity of BotFlowMon is $O(n)$, where n is the total number of input NetFlow records. The detailed analysis is in Appendix D.

4.7.2 *Performance*

We tested BotFlowMon's performance to see how fast it is in detecting social bot traffic. We tested it on a personal laptop with a quad-core 2.7-GHz CPU, 16-GB memory, but

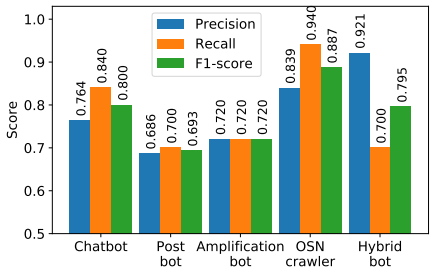


Fig. 19. Multiclass classification scores for different categories of social bots.

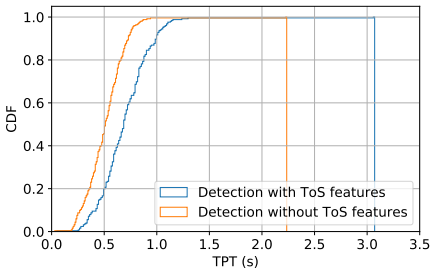


Fig. 20. Cumulative distribution functions of TPT (time per transaction).

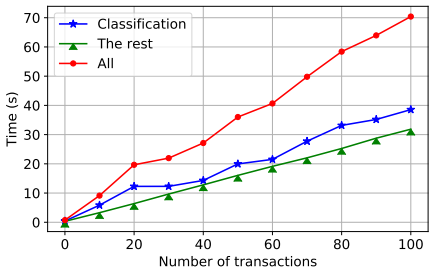


Fig. 21. Performance of BotFlowMon and its modules (using CNN with ToS features).

no GPU. We ran it in a single thread with the worst case for performance where *every* flow is an OSN flow.

We first evaluated how long it takes for BotFlowMon to process a single transaction, or **TPT (time per transaction)**. Note among the three factors we analyzed in Section 4.4, only the ToS features will affect TPT. We thus measured BotFlowMon's TPT with both 6×200 and 4×200 transaction fingerprints. Figure 20 shows the cumulative distribution functions (CDFs) of the TPT for both. Clearly, the TPT with 4x200 transaction fingerprints is shorter, where TPTs with 6×200 and 4×200 transaction fingerprints are 0.71 and 0.49 seconds on average, respectively.

Moreover, we measured the time spent on each module of BotFlowMon and found that about half time of a TPT is spent on the classification step. For the TPT with 6×200 transaction fingerprints, its 56.75% is from the final classification step and its 43.25% is from the rest operations.

Finally, Figure 21 shows that as the number of transactions increases, the total processing time increases basically linearly. Here, the processing time is the total time for the classification and the rest operations. The linear increase of the classification time is less stable than that of the rest operations, as the classification workload for each transaction depends on the number of its actions, which is uncertain.

Clearly, there is an ample space to improve both hardware and software support (e.g., using GPU and multithreading) for running BotFlowMon. Even without doing so, BotFlowMon is already fairly fast to detect social bot traffic in real time.

5 Limitations and Open Issues

A primary contribution of BotFlowMon is to use data from network layers 3 and 4 to detect anomalies at network layer 7. However, BotFlowMon has some limitations:

1) As BotFlowMon is focused on distinguishing the social bot traffic from the real user traffic, its ability to identify different categories of social bots is not ideal (demonstrated in Section 4.6). To address this limitation, we could consider using social bot traffic identified by BotFlowMon to trace relevant OSN content and then leveraging the content to help classify social bots.
2) In fact, not all the social bots are necessarily malicious, and the boundary between "good" bots and "bad" bots can be blurry. To distinguish social bots with malicious intentions from innocent ones, not only will we very likely need content data of these social bots, but we also will have to apply techniques such as natural language processing to identify the intention behind a social bot.
3) If the training data of BotFlowMon does not include traffic of certain social bots, such as zero-day social bots, the system probably will not be able to detect them.

BotFlowMon may also be enhanced with new functions and capabilities:

1) We may explore adding more features to BotFlowMon to improve its accuracy, such as the AS number of an IP, non-OSN traffic from the same IP, and time of the day.
2) We may add an online learning capability to BotFlowMon to make it adaptable to possible concept drifts.
3) We may study how to enhance BotFlowMon with adversarial machine learning techniques such that it is resilient against social bots that try to mimic real users.

6 Conclusions

Today's social bots are becoming far more sophisticated and threatening than before. To prevent the online social ecosystems from being troubled or attacked by them, this paper proposes a social bot detection method called BotFlowMon, which takes advantage of big networking data to identify the traffic of OSN bots. As the networking data it processes are only information from network layers two and three and contain nothing about OSN content and structure, BotFlowMon departs from previous content-based, structure-based, and crowdsourcing-based solutions and is a content-agnostic, privacy-preserving, and efficient approach.

BotFlowMon devises several new techniques and algorithms, including an aggregation technique that derives transaction datasets from network flow records, a data fusion technique that extracts features from transactions datasets, and an innovative density-valley-based clustering algorithm that can both divide a transaction fingerprint into multiple actions and serve as a general clustering algorithm like DBSCAN to process other types of data. It can identify traffic flows from social bots with an accuracy of around 95%. It is also easy to deploy; any Internet service provider or enterprise networks, as long as they are able to access the traffic flow records such as NetFlow, can deploy BotFlowMon. Finally, it can monitor traffic of a network and detect social bots in real time; the study of the worst case on a testing machine shows BotFlowMon can determine whether a flow is from a social bot or not within 0.71 seconds on average after it sees the flow.

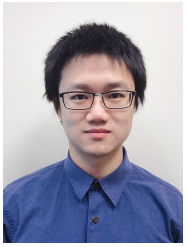
ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1564348. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

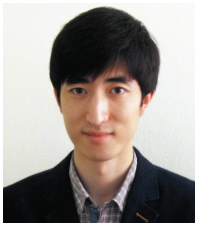
[1] A. Janowitz, "Statista: The statistics portal for market data, market research and market," https://www.statista.com, 2019.
[2] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The rise of social bots," *Communications of the ACM*, vol. 59, no. 7, pp. 96–104, 2016.
[3] J. Zhang, R. Zhang, Y. Zhang, and G. Yan, "The rise of social botnets: Attacks and countermeasures," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 1068–1082, 2016.
[4] E. Ferrara, "Manipulation and abuse on social media," *ACM SIGWEB Newsletter*, no. Spring, pp. 1–9, 2015.
[5] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, "All your contacts are belong to us: Automated identity theft attacks on social networks," in *18th international conference on world wide web*, 2009.
[6] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and characterizing social spam campaigns," in *10th ACM Internet measurement conference*, 2010, pp. 35–47.
[7] "Attackers use large-scale bots to launch attacks on social media platforms," https://www.helpnetsecurity.com/2019/08/27/attacks-on-social-media-platforms/, 2019, Help Net Security.
[8] B. Claise, "Cisco Systems NetFlow services export version 9," RFC 3954, IETF, 2004.
[9] R. Sommer and A. Feldmann, "NetFlow: Information loss or win?" in *2nd ACM workshop on Internet measurment*, 2002, pp. 173–174.
[10] Y. Feng, J. Li, L. Jiao, and X. Wu, "BotFlowMon: Learning-based, content-agnostic identification of social bot traffic flows," in *IEEE conference on communications and network security*, 2019.
[11] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, "Detecting automation of Twitter accounts: Are you a human, bot, or cyborg?" *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 6, pp. 811–824, 2012.
[12] K. Thomas, C. Grier, and V. Paxson, "Adapting social spam infrastructure for political censorship," in *5th USENIX workshop on large-scale exploits and emergent threats*, 2012.
[13] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai, "Uncovering social network Sybils in the wild," *ACM Transactions on Knowledge Discovery from Data*, vol. 8, no. 1, pp. 1–29, 2014.
[14] G. Wang, T. Konolige, C. Wilson, X. Wang, H. Zheng, and B. Y. Zhao, "You are how you click: Clickstream analysis for Sybil detection," in *USENIX Security*, 2013, pp. 241–256.
[15] Q. Cao and X. Yang, "SybilFence: Improving social-graph-based Sybil defenses with user negative feedback," *arXiv preprint arXiv:1304.3819*, 2013.
[16] N. Z. Gong, M. Frank, and P. Mittal, "SybilBelief: A semi-supervised learning approach for structure-based Sybil detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 6, pp. 976–987, 2014.
[17] J. Jia, B. Wang, and N. Z. Gong, "Random walk based fake account detection in online social networks," in *47th annual IEEE/IFIP international conference on dependable systems and networks*, 2017, pp. 273–284.
[18] B. Wang, N. Z. Gong, and H. Fu, "GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs," in *IEEE international conference on data mining*, 2017, pp. 465–474.
[19] G. Wang, M. Mohanlal, C. Wilson, X. Wang, M. Metzger, H. Zheng, and B. Y. Zhao, "Social turing tests: Crowdsourcing Sybil detection," in *The network and distributed system security symposium*, 2013.
[20] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux, "Mechanical cheat: Spamming schemes and adversarial techniques on crowdsourcing platforms," in *CrowdSearch*, 2012, pp. 26–30.
[21] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao, "Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers," in *USENIX security symposium*, 2014, pp. 239–254.
[22] J. Ratkiewicz, M. D. Conover, M. Meiss, B. Gonçalves, A. Flammini, and F. M. Menczer, "Detecting and tracking political abuse in social media," in *Fifth international AAAI conference on weblogs and social media*, 2011.
[23] A. Karataş and S. Şahin, "A review on social bot detection techniques and research directions," in *International security and cryptology conference*, 2017, pp. 156–161.
[24] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song, "Design and evaluation of a real-time URL spam filtering service," in *IEEE symposium on security and privacy*, 2011, pp. 447–462.
[25] S. Yardi, D. Romero, G. Schoenebeck *et al.*, "Detecting spam in a Twitter network," *First Monday*, 2010.
[26] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *26th annual computer security applications conference*, 2010, pp. 1–9.
[27] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on Twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference*, 2010, p. 12.
[28] K. Lee, B. D. Eoff, and J. Caverlee, "Seven months with the devils: A long-term study of content polluters on Twitter," in *Fifth international AAAI conference on weblogs and social media*, 2011.
[29] A. H. Wang, "Don't follow me: Spam detection in Twitter," in *International conference on security and cryptography*, 2010, pp. 1–10.
[30] K. Thomas, C. Grier, D. Song, and V. Paxson, "Suspended accounts in retrospect: An analysis of Twitter spam," in *ACM Internet measurement conference*, 2011, pp. 243–258.
[31] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer, "Botornot: A system to evaluate social bots," in *25th international conference companion on world wide web*, 2016, pp. 273–274.
[32] J. Song, S. Lee, and J. Kim, "Spam filtering in Twitter using sender-receiver relationship," in *International workshop on recent advances in intrusion detection*, 2011, pp. 301–317.
[33] G. Danezis and P. Mittal, "SybilInfer: Detecting Sybil nodes using social networks," in *The network and distributed system security symposium*, 2009, pp. 1–15.
[34] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, "Aiding the detection of fake accounts in large scale social online services," in *9th USENIX conference on networked systems design and implementation*, 2012, pp. 197–210.
[35] B. Wang, J. Jia, and N. Z. Gong, "Graph-based security and privacy analytics via collective classification with joint weight learning and propagation," in *The network and distributed system security symposium*, 2019.
[36] B. Wang, L. Zhang, and N. Z. Gong, "SybilSCAR: Sybil detection in online social networks via local rule based propagation," in *IEEE conference on computer communications*, 2017, pp. 1–9.
[37] P. Gao, B. Wang, N. Z. Gong, S. R. Kulkarni, K. Thomas, and P. Mittal, "SybilFuse: Combining local attributes with global structure to perform robust Sybil detection," in *IEEE conference on communications and network security*, 2018, pp. 1–9.
[38] D. Koll, J. Li, J. Stein, and X. Fu, "On the state of OSN-based Sybil defenses," in *IFIP networking*, 2014, pp. 1–9.
[39] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5–16, 2007.
[40] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for Internet traffic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, 2007.
[41] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *4th ACM Internet measurement conference*, 2004, pp. 135–148.
[42] D. Rossi and S. Valenti, "Fine-grained traffic classification with NetFlow data," in *6th International wireless communications and mobile computing conference*, 2010, pp. 479–483.
[43] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight application classification for network management," in *ACM SIGCOMM workshop on Internet network management*, 2007.
[44] A. Bashir, C. Huang, B. Nandy, and N. Seddigh, "Classifying P2P activity in NetFlow records: A case study on BitTorrent," in *IEEE international conference on communications*, 2013, pp. 3018–3023.
[45] M. H. Bhuyan, H. J. Kashyap, D. K. Bhattacharyya, and J. K. Kalita, "Detecting distributed denial of service attacks: methods, tools and future directions," *The Computer Journal*, vol. 57, no. 4, pp. 537–556, 2014.
[46] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *Third international conference on emerging security information, systems and technologies*, 2009, pp. 268–273.
[47] P. Li, M. Salour, and X. Su, "A survey of internet worm detection and containment," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 20–35, 2008.

[48] Y. Feng, D. Sisodia, and J. Li, "Poster: Content-agnostic identification of cryptojacking in network traffic," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, p. 907–909.
[49] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *8th international conference on emerging networking experiments and technologies*, 2012, pp. 349–360.
[50] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *28th annual computer security applications conference*, 2012, pp. 129–138.
[51] G. Gu, R. Perdisci, J. Zhang, W. Lee *et al.*, "BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection," in *USENIX security symposium*, 2008, pp. 139–154.
[52] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in DNS traffic," in *7th IEEE international conference on computer and information technology*, 2007, pp. 715–720.
[53] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A method for monitoring traffic in switched and routed networks," RFC 3176, IETF, 2001.
[54] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "BGPStream: a software framework for live and historical BGP data analysis," in *ACM Internet measurement conference*, 2016, pp. 429–444.
[55] T. Hayashi and T. Miyazaki, "High-speed table lookup engine for IPv6 longest prefix match," in *IEEE global telecommunications conference*, 1999, pp. 1576–1581.
[56] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge Discovery and Data Mining*, 1996, pp. 226–231.
[57] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.
[58] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[59] K. Hornik, M. Stinchcombe, H. White *et al.*, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
[60] "Keras," https://keras.io, 2015.
[61] "TensorFlow: Large-scale machine learning on heterogeneous systems," https://www.tensorflow.org/, 2015.
[62] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, 2009, pp. 35–48.
[63] H. Gao, Y. Chen, K. Lee, D. Palsetia, and A. N. Choudhary, "Towards online spam filtering in social networks." in *The network and distributed system security symposium*, 2012, pp. 1–16.
[64] R. S. Wallace, *Be Your Own Botmaster: The Step By Step Guide to Creating, Hosting and Selling Your Own AI Chat Bot On Pandorabots*. ALICE AI foundations, Incorporated, 2003.
[65] H. Al-Zubaide and A. A. Issa, "Ontbot: Ontology based chatbot," in *Fourth international symposium on innovation in information & communication technology*, 2011, pp. 7–12.
[66] "The open source chatbot and artificial intelligence platform," 2020. [Online]. Available: http://www.botlibre.org/
[67] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: The underground on 140 characters or less," in *17th ACM conference on computer and communications security*, 2010, pp. 27–37.
[68] "Poster bots from GitHub," https://github.com/search?q=poster+bot, 2018.
[69] J. Roesslein, "Tweepy documentation," http://docs.tweepy.org/en/latest/, 2009.
[70] W. Graham, *Facebook API developers guide*. Infobase Publishing, 2008.
[71] "Twitter bot made with Golang," https://github.com/benoitletondor/TwitterBot, 2020.
[72] D. Hardt, "The OAuth 2.0 authorization framework," 2012.
[73] "Facebook-Cambridge Analytica data scandal," https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal, 2019.



Yebo Feng is a Ph.D. student in the Department of Computer and Information Science at the University of Oregon (UO), where he conducts his research in the Center for Cyber Security and Privacy. He received his M.S. degree from UO in 2018, B.E. from Yangzhou University in 2016, all in computer science. His research interests include computer security, anomaly detection, DDoS defense, and data analysis. He is the recipient of the Best Paper Award of 2019 IEEE CNS, Gurdeep Pall Graduate Student Fellowship of UO, and Ripple Research Fellowship. He has served as the reviewer of IEEE TDSC, ACM TKDD, and IEEE JSAC.

Jun Li is a Professor in the Department of Computer and Information Science and founding director of the Center for Cyber Security and Privacy at the University of Oregon. He received his Ph.D. from UCLA in 2002 (with Outstanding Doctor of Philosophy honor), M.E. from Chinese Academy of Sciences in 1995 (with Presidential Scholarship), and B.S. from Peking University in 1992, all in computer science. His research is focused on networking, distributed systems, and network security, with about 90 peer-reviewed publications. He has served on US National Science Foundation research panels and more than 70 international technical program committees, including chairing six of them. He is a senior member of ACM and IEEE and an NSF CAREER awardee in 2007.



Lei Jiao received the Ph.D. degree in computer science from University of Göttingen, Germany. He is currently an assistant professor at the Department of Computer and Information Science, University of Oregon, USA. Previously he worked as a member of technical staff at Alcatel-Lucent/Nokia Bell Labs in Dublin, Ireland and also as a researcher at IBM Research in Beijing, China. He is interested in exploring optimization, control, learning, mechanism design, and game theory to manage and orchestrate large-scale distributed computing and communication infrastructures, services, and applications. He has published papers in journals such as JSAC, TON, TMC, and TPDS, and in conferences such as MOBIHOC, INFOCOM, ICNP, ICDCS, SECON, and IPDPS. He served as a guest editor for IEEE JSAC Series on Network Softwarization and Enablers. He was on the program committees of many conferences including MOBIHOC, INFOCOM, ICDCS, and IWQoS, and was the program chair of multiple workshops with INFOCOM and ICDCS. He was also a recipient of the Best Paper Awards of IEEE CNS 2019 and IEEE LANMAN 2013, and the 2016 Alcatel-Lucent Bell Labs UK and Ireland Recognition Award.

Xintao Wu is a Professor and the Charles D. Morgan/Acxiom Endowed Graduate Research Chair in Database in the Department of Computer Science and Computer Engineering at the University of Arkansas. He held a faculty position at the College of Computing and Informatics at the University of North Carolina at Charlotte from 2001 to 2014. He got his Ph.D. degree in Information Technology from George Mason University in 2001. He received his BS degree in Information Science from the University of Science and Technology of China in 1994, an ME degree in Computer Engineering from the Chinese Academy of Space Technology in 1997. His major research interests include data mining, data privacy, fraud detection, fairness aware machine learning, and big data analytics. He is a member of ACM and IEEE and an NSF CAREER Awardee in 2006.

APPENDIX A
EXTENSION FOR THE DENSITY-VALLEY-BASED CLUSTERING

Algorithm 1 only applies to one-dimension datasets since the valley point competition mechanism only handles circumstances where two clusters share a valley point. This design is sufficient for processing transaction fingerprint data, but not able to deal with multi-dimension datasets where there can be more than two clusters sharing a valley point. We thus designed a more universal valley point competition mechanism in Algorithm 2 to solve this problem. In particular, if data point e in Algorithm 1 is a valley point of *multiple* clusters $\{c_1, c_2, ..., c_n\}$, it can invoke Algorithm 2 instead. It first divides competing clusters into a low-density group and a high-density group. It then merges all the clusters in the low-density group, as well as the cluster with the lowest density in the high-density group, into one cluster, and assigns valley point e to the cluster with the fewest data points among all clusters. With the help of Algorithm 2, Algorithm 1 can handle more complicated transaction fingerprint configurations, or even serve as a general clustering algorithm to process other types of high-dimension and low-dimension datasets.

Algorithm 2 Valley point competition for $n \geq 2$ clusters.

Input: Valley point e, set of competing clusters $C_n = \{c_1, c_2, ..., c_n\}$, set C from Algorithm 1 that stores current clusters, valley point index ρ
$C_{low} = \phi$; $C_{high} = \phi$ ▷ two sets to store the low density and high density clusters from C_n, respectively
$C = C - C_n$ ▷ remove C_n from C
for cluster c in C_n **do**
  if $c.density \cdot \rho \leq e.density$ **then**
    Add c to C_{low}
  else
    Add c to C_{high}
  end if
end for
if $C_{low} \neq \phi$ **then**
  if $|C_{high}| >= 2$ **then**
    c_{border} = the cluster with the lowest density in C_{high}
    remove c_{border} from C_{high}
    $c_{new} = merge(C_{low}, \{c_{border}\})$
    $C_{all} = C_{high} \cup \{c_{new}\}$
    Add e to the cluster with the fewest data points in C_{all}
    Add C_{all} to C
  else
    $c_{new} = merge(C_{low}, C_{high})$
    Add e to cluster c_{new}
    Add c_{new} to C
  end if
else
  Add e to the cluster with the fewest data points in C_{high}
  Add C_{high} to C
end if
return C

APPENDIX B
SOCIAL BOTS SIMULATION

There are various types of social bots existing in the world, also, they can create different traffic flows during operations. BotFlowMon's design requires inputting a significant amount of labeled traffic data as ground truth. To derive the labeled dataset, we categorized the social bots into the following five types and comprehensively simulated them.

B.1 Chatbot

Chatbots are active in messaging applications such as Twitter Direct Messages, Facebook Messenger, or WeChat. Either artificial-intelligence-powered or merely logic-based, they can automatically perform conversations with regular users.

The simulation of chatbots relies on existing widely used chatbot frameworks, APIs, and open-source programs such as botmaster [64], Ontbot [65], BotLibre [66], and python-twitter API. We created many Twitter and Facebook bot accounts only for research purposes under the OSNs' terms and conditions of service. The chatbots we created only talked to the recruited participants during the data generation process. We collected multitudinous traffic flows of the conversations between real users and the chatbots, with different frequencies of interaction, response time and transmission content (image, audio, text, or hyperlink).

B.2 Post Bot

In part due to the easily usable official and third-party APIs, poster bots have become the most common social bots in OSN. They distribute spam tweets and Facebook posts with malicious URLs in most cases or malicious texts occasionally [3] [67].

In order to simulate post bots, we downloaded several popular open-source poster bot software from GitHub [68] and wrote some poster bot programs based on APIs such as Tweepy [69] and Facebook API [70]. The open-source programs we used included botmaster [64], BotLibre [66], and TwitterBot [71]. We first set up OSN accounts on Twitter and Facebook platforms. Then, we ran these bots programs with different frequencies, including random frequencies, during different time periods to post some harmless messages that contained texts, videos, images and external URLs on Twitter and Facebook.

B.3 Amplification Bot

Amplification bot, due to the large volume of messages it can generate, is often used to create hot topics for commercial promotion, consensus manipulation, and spam distribution. Without creating new content, amplification bots often work as fake followers, such as those Twitter or Facebook accounts specifically created to inflate the number of followers of a target account. Amplification bots also can serve as forwarding and liking robots, popularize junk information, and help commercial promotion.

Since the social relationship is unknown in NetFlow data, we only need to simulate every individual amplification bot's interactions with OSNs. The simulation of amplification bots is similar to the post bot simulation. We used open-sourced programs [64], [66], [71] and implemented

API-based bot scripts to simulate amplification bots. Besides, we used OAuth [72] software for token management and switching accounts.

B.4 OSN Crawler

OSN crawlers can exploit OSNs to aggregate data of a large number of OSN users for re-publication or other nefarious purposes that violate users' privacy and security. There are two types of OSN crawlers and we simulated both of them. One is API-based, which relies on a large botnet to extract users' private, sensitive data. Twitter and Facebook used to have powerful APIs such that even a small number of accounts can fetch a large amount of information from their sites. Recently OSN operators have taken plenty of measures to limit the APIs' capacity in crawling user data, especially after the Facebook-Cambridge Analytica data scandal [73]. Moreover, in OSNs it is common that a user's information can only be accessed by their friends or followers. Therefore, a significant amount of social bots are necessary to overcome the API limitations and retrieve users' private information. The following code exemplifies a crawling process.

```
import twitter
api = twitter.Api()
api.GetUser(user)
api.GetFollowers()
api.GetStatus(status_id)
api.GetFriends(user)
api.LookupFriendship(user)
```

Another type of OSN crawler is page crawler. Instead of using API, it reads the HTML files of OSNs and uses regular expressions to extract target information. The NetFlow traffic of this bot has a strong resemblance to a regular user's traffic but still differs in key aspects, such as flow density and operation frequency.

B.5 Hybrid Bot

The hybrid bot is not a specific type of social bot. Instead, it is an arbitrary combination of different types of social bots. This characteristic makes the activities of a hybrid bot hard to detect. However, with BotFlowMon's subdivision module, we can subdivide its transactions into actions, thus still able to detect it.

We implemented a hybrid bot that consists of a chatbot, a post bot, an amplification bot, and an OSN crawler on a single node. During the execution of the hybrid bot, its chatbot is always active, so it automatically replies whenever other accounts send it messages. Meanwhile, it randomly conducts actions provided by other underlying bots. We simulated its activities with both periodic and random frequencies.

APPENDIX C
ETHICAL AND HUMAN SUBJECT ISSUES

We carefully addressed the ethical and human subject issues involved in this research. As the development and evaluation of BotFlowMon involved human subjects, we obtained the Institutional Review Board (IRB) approval to ensure that our procedures are ethical and appropriate in order to safeguard the welfare of the campus network and the privacy of research participants. We introduce the main measures we adopted to address the ethical considerations below.

C.1 Data Generation

In recruiting participants, we adopted an informed consent procedure, which provided the potential participants with a clear description about the project before they agree. After the data generation, all their personal data were deleted immediately. In simulating social bots, we ensured they followed the terms of service of OSN providers, were harmless to other users and the OSN environments, and left no traces after each experiment. For example, we used weather reports, university announcements, and encyclopedic knowledge to simulate fraud and spam posts for post and amplification bots. The external URLs contained in the bot messages were well-known innocent URLs, such as google.com and facebook.com. The chatbots and OSN crawlers only interacted with the informed participants, and the personal data gathered by them were not stored in any form.

C.2 Data Collection

All the data we collected were content-agnostic NetFlow records, along with the labels, time stamps, and short descriptions of real user and social bot activities. No payload or content data were downloaded and stored during the process. For the NetFlow records from the campus network, every internal IP address was anonymized, so it cannot be mapped back to any individual. Using anonymous IP addresses, however, makes it hard to identify the NetFlow records created by ourselves. To address this issue, every time we generated legitimate or illegitimate traffic, we pinged several external IP addresses and recorded their time stamps to create distinctive features of these traffic, making it easy to identify their NetFlow records and discard all irrelevant ones. In addition, all the data we collected were stored in an encrypted form, and any access to the data must have the approval of the principal investigator of this project.

APPENDIX D
TIME COMPLEXITY ANALYSIS

D.1 Time Complexities for Different Modules

Preprocessing module: This module's input is all the NetFlow records from a monitored network. Assume their number is n. This module applies the longest prefix match to output OSN-related NetFlow records, which takes a linear time. Its time complexity is therefore $O(n)$.

Flow aggregation module: This module's input is OSN-related NetFlow records per OSN account (which can be either a real user or a social bot). Assume the number of time bins of these records is m. As this module uses DBSCAN to aggregate related NetFlow records into transactions, it takes $O(mlogm)$ time.

Transaction fingerprint generation module: This module's input is a set of NetFlow records that form a single transaction. Its purpose is to produce an $f \times N$ matrix as the fingerprint of the transaction. This module's time complexity is thus $O(fN)$.

Transaction subdivision module: This module's input is the $f \times N$ fingerprint matrix of a single transaction. It applies its density-valley-based clustering algorithm to subdivide the transaction and output a set of actions in the form of action fingerprint matrixes. As the time complexity of the clustering algorithm is similar to DBSCAN, this module's time complexity is $O(NlogN)$.

Machine learning & classification module: This module's input is j actions that a transaction is subdivided into. Assuming the number of layers in CNN or MLP classification model is l, this module is to classify each action with a time complexity $O(l)$. This module's time complexity is thus $O(lj)$.

D.2 Overall Time Complexity

As BotFlowMon processes incoming NetFlow records online, its time complexity is the sum of the time complexity of every module. Assume after the preprocessing module the selected NetFlow records belong to a OSN accounts. Also assume the flow aggregation module outputs b transactions per OSN account. BotFlowMon's time complexity is thus $O(n) + a{\cdot}O(mlogm) + a{\cdot}b{\cdot}O(fN) + a{\cdot}b{\cdot}O(NlogN) + a{\cdot}b{\cdot}O(lj)$. Here, m is a constant as BotFlowMon receives NetFlow records periodically, f and N are constants as f is 4 or 6 and N is 200 in our setup, and l is a constant with a default value of 9 for CNN and 7 for MLP. So BotFlowMon's time complexity can be simplified as $O(n)+O(a)+O(ab)+O(ab)+O(abj)$, which is equivalent to $O(n)+O(abj)$. Note abj represents all the actions by all OSN accounts from these n NetFlow records, which is approximately proportional to n. Therefore, BotFlowMon's time complexity is $O(n)$, where n is the total number of input NetFlow records.