BotFlowMon: Learning-Based, Content-Agnostic Identification of Social Bot Traffic Flows

Yebo Feng, Jun Li, Lei Jiao
Computer and Information Science
University of Oregon
Eugene, USA
{yebof, lijun, jiao}@cs.uoregon.edu

Xintao Wu
Computer Science & Computer Engineering
University of Arkansas
Fayetteville, USA
xintaowu@uark.edu

***Abstract*—With the fast-growing popularity of online social networks (OSN), maintaining the security of OSN ecosystems becomes essential for the public. Among all the security threats facing OSN, malicious social bots have become the most common and detrimental. These bot programs 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 bot detection approaches that have strict requirements on data sources (e.g., private payload information, social relationships, or activity histories), this paper proposes a detection method called BotFlowMon that relies only on NetFlow data as input to identify OSN bot traffic, where every NetFlow record is a summary of a traffic flow on the Internet and contains no payload content. BotFlowMon introduces several new algorithms and techniques to help use machine learning to classify the social bot traffic from the real OSN user traffic, including aggregating NetFlow records to obtain transaction data, fusing transaction data to extract features and visualize flows, as well as subdividing transactions into basic actions. Our evaluation shows that with 535GB raw NetFlow records as input, BotFlowMon can efficiently classify the traffic from social bots, including chatbot, amplification bot, post bot, crawler bot, and hybrid bot, with 92.33–93.61% accuracy.

***Index Terms*—online social network (OSN), OSN security, social bot, NetFlow data, OSN bot traffic, machine learning**

I. Introduction

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

Existing approaches to detecting social bots need to utilize the social relationship topology, private content data from user posts or messages, or OSN account activity histories, 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 traffic flow information as input, such as NetFlow records [8], to differentiate the social bot traffic from the real user traffic. As the NetFlow records are low-volume, coarse-grained summaries of traffic flows and contain *no* payload information [9], our approach is privacy-preserving and can be deployed by Internet/network service providers. We show that even without traffic payload, BotFlowMon can accurately and quickly detect the social bot traffic.

BotFlowMon harnesses the power of machine learning on big data for the best efficacy in classifying social bot traffic, in order to label NetFlow data that encompass social bot traffic versus real user traffic. BotFlowMon employs five modules:

1) The *preprocessing module* that filters out noises and irrelevant data from raw traffic flow 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 newly designed 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-and-valley-based clustering algorithm to further divide each transaction into multiple actions, thus reducing the training data volume requirement and accelerating the convergence of training;
5) The *machine learning & classification module* that takes the action-level data as input to construct a transaction-level social bot classification model with multilayer perceptron (MLP) and convolutional neural network (CNN).

Our evaluation shows that BotFlowMon can be easily deployed by Internet service providers or enterprise networks and it can process big networking data with high efficiency and detect social bot flows with high accuracy.

After preprocessing 535 gigabytes of raw NetFlow data, BotFlowMon processed 30,932,991 NetFlow records and achieved an accuracy between 92.33% and 93.61%. This accuracy provides a solid starting point for network operators

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.

978-1-5386-7117-7/19/$31.00 ©2019 IEEE

to take further actions. Simultaneously, the conciseness of NetFlow data provides an advantage for fast and efficient data processing; even only running on a laptop with 2.7-Ghz CPU and 16-GB memory, BotFlowMon can support real-time social bot detection for a campus-level network.

II. Related Work

In order to maintain a secure and privacy-respecting online social environment for users, various approaches have been developed to identify social bot programs. According to the input of these approaches, they are often either content-based or OSN-topology-based, with a new trend on using crowdsourcing techniques to discover bots. None of these approaches inspect network traffic to identify the traffic flows from social bots. On the other hand, there are some work on detecting traffic of other types of bots, such as DDoS bots.

The key idea of content-based bot detection is to leverage the differences between bot programs and real users in terms of OSN post syntax, content, account activity histories, and post linguistic features. As OSN providers can easily obtain a massive amount of OSN user data nowadays, many content-based detection approaches have been proposed [10]–[13], which apply machine learning, natural language processing, semantic analysis, and so on, to construct a classification model to bots. For example, "BotOrNot" [10], the first social bot detection framework publicly available for Twitter, analyzed content of 15,000 manually verified social bots and 16,000 real user accounts and achieved 86% accuracy. In general, content-based approaches require a large volume of labeled social content data and incur severe privacy concerns.

Approaches based on topology (i.e., social network structure) focus on detecting amplification bots and Sybil accounts, where a bot master controls multiple accounts to perform similar actions in an organized manner. Such an approach usually obtains the topology structure of an online social network first, and then applies methods such as Random Walk, Bayesian Network, and Loopy Belief Propagation to identify malicious accounts [14]–[18].

A new trend in social bot detection is to use crowdsourcing [19]–[21] in which one can ask individual participants to judge whether a program is a bot or not and then aggregate the decisions from all participants. For example, research in [22] relies on a crowdsourcing layer to have individual users to determine whether an OSN account is a bot account or not, and further a filtering layer to filter out unsatisfactory report from individual users. However, a crowdsourcing approach tends to incur a long running time, a high cost, and privacy risk. In addition, a crowdsourcing approach is hardly applicable to identifying social bot traffic flows in a given network, since it is almost impossible for a crowd of participants all over the Internet to even access such traffic flows.

Researchers have studied how to identify traffic generated by other types of bots. For example, research in BotMiner [23] investigated the detection of Internet Relay Chat (IRC) protocol-based bot traffic; BotFinder [24] uses machine learning to identify the key features of command-and-control (C&C) communications of botnets; Research in [25] identifies anomalous DNS traffic generated by bot machines. Unfortunately, these methods are not applicable to detecting the traffic flows from social bots.

III. BotFlowMon Design

A. Overview

In order to detect if any machine in its network is a social bot and producing social bot traffic, a network service provider can deploy BotFlowMon on any router that sits between its network and OSN servers, so long as it can access the traffic between its network and OSN servers. It imports the traffic flow data from the router to distinguish social bot traffic flows from real OSN user flows.

There are different traffic flow formats. We focus on the widely used NetFlow [8] format in this paper; our design can easily extend to other flow formats such as sFlow [26].

The information to leverage from NetFlow records is simple and straightforward. Every NetFlow record only contains information derived from the header of IP packets of the same flow, such as timestamps, IP addresses, port number, and packets per second. Of a particular note here is that a NetFlow record contains *no* payload data of any IP packet.

Figure 1 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 describe each module in detail below.

B. Preprocessing

We preprocess the raw NetFlow data collected from a router as follows.

We first extract the traffic flows only related to OSNs. After removing flows with zero bytes, zero duration, or irrelevant protocols (such as ICMP), BotFlowMon will check each NetFlow record's source or destination IP address to verify whether or not the IP address is associated with the social network site(s) in question (e.g., Facebook or Twitter). BotFlowMon then discards those NetFlow records that do not match. One issue here is that each OSN site may own hundreds or even more IP prefixes and the current IP prefixes of an OSN site may change over time. We leverage BGP stream [27] to obtain the list of current and historical IP prefixes of an OSN site, and then look up which IP prefixes are associated with an OSN site at a different time. In order to deal with a large number of flows efficiently, the matching process we designed here 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 discard flows not related to the OSN site(s) in question in real time.

We also group network flows by OSN users. Here, a user is determined by one unique combination value of IP address and port number. In particular, we use source IP and port to

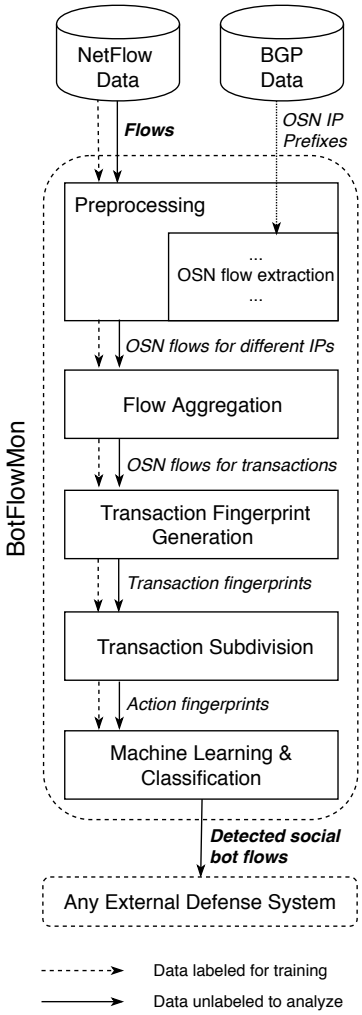


Fig. 1. BotFlowMon architecture. It has two modes of operation: training (data in dashed line) and monitoring (data in solid line).

identify users for outgoing flows, and use destination IP and port to identify users for incoming flows.

C. Flow Aggregation

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 and sometimes even identical. We thus introduce the flow aggregation module in BotFlowMon to inspect *collective* OSN behaviors of flows and capture distinct patterns of social bot versus real user behaviors.

The flow aggregation module aggregates all the NetFlow records generated by the same **transactions**, so that we can inspect and compare transactions of a social bot against OSN

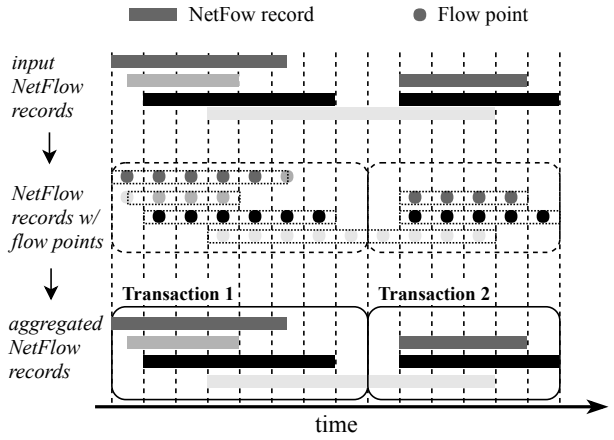


Fig. 2. A flow aggregation example using modified DBSCAN. Every time bin is 0.1 seconds. Six NetFlow records are clustered into two transactions by converting them to NetFlow records with flow points. (Each flow point has a different gray level, with a darker gray indicating a higher traffic volume.) One NetFlow record was fragmented into two records, each at a different transaction.

transactions of real users, including defining and comparing features with regard to the collective OSN behaviors of flows. 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, a transaction 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 [28] to aggregate/cluster flows into multiple different transactions, with each transaction composed of multiple flows. 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 time bin. Every flow point has a traffic volume derived from the bits per second (bps) of the flow multiplied by the length of the time bin.
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.

Figure 2 shows an example of this procedure.

D. Transaction Fingerprint Generation

With multiple transactions obtained from the flow aggregation module, BotFlowMon now generates a fingerprint for each transaction to make different transactions directly comparable to each other. Our basic idea is a data fusion method that

TABLE I
6xN 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_{i2}	...	$bps^i_{t_N}$
5: incoming pps	pps^i_{t1}	pps^i_{i2}	...	$pps^i_{t_N}$
6: incoming ToS	tos^i_{t1}	tos^i_{i2}	...	$tos^i_{t_N}$

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 of all flows in the transaction to the latest end time of all flows in the transaction, and f is the number of features of the transaction over each time bin.

Table I shows a 6xN example transaction fingerprint matrix. Row 1 to row 3 are features extracted from outgoing flows and row 4 to row 6 are features extracted from incoming flows. Both use bits per second (**bps**), packets per second (**pps**) and type of service (**ToS**) as features. Note at any time bin there can be more than one flows 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 that are active in that time bin. The outgoing or incoming ToS feature for that time bin, however, is not numerical, and its value is the ToS value of the flow that has the largest bps value during that time bin. However, because the usage of ToS field in datagrams has not been standardized and different ISPs may further modify the ToS fields when collecting NetFlow records, the ToS feature may not be reliable to help produce a transaction fingerprint. As such, we 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. Using the 6xN matrix from table I as an example, we can map every bps, pps and ToS value in the matrix to a number between 0 and 255, which can be done by analyzing the dispersion of bps, pps and ToS values from a very large data set of NetFlow records. We draw three quantile-quantile plots that shows the dispersion of bps, pps and ToS values based on 235-GB NetFlow data of campus traffic, which led to three functions $f_r(bps)$, $f_g(pps)$, and $f_b(tos)$, respectively, to normalize the bps, pps, and ToS values to numbers between 0 and 255.

We can also easily visualize a transaction fingerprint matrix for inspection and analysis, since all the values lie between 0 and 255. For each transaction, BotFlowMon 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. Figure 3 is an example of visualizing a transaction fingerprint, which has 220 flows and represents a real user spending 35.74 seconds browsing Facebook.

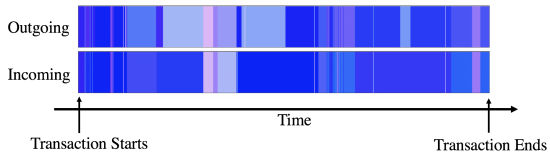


Fig. 3. Visualizing the fingerprint of a transaction lasting 35.74 seconds with 220 flows.

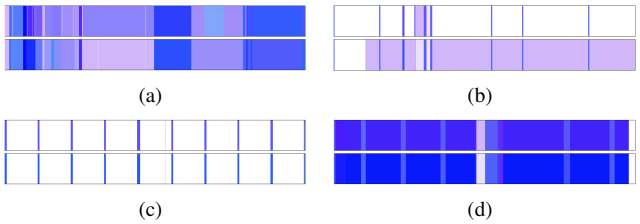


Fig. 4. More transaction fingerprint examples

Figure 4 shows more examples of visualized transaction fingerprints from labeled ground truth. Transaction fingerprints in Figures 4(a) and 4(b) are two Twitter users messaging each other through Twitter Direct Messages (DM), where the transaction fingerprint in Figure 4(a) is generated by a real user and the transaction fingerprint in Figure 4(b) is generated by a chatbot. The transaction fingerprint in Figure 4(c) is created by a social bot that uses APIs to tweet text messages on Twitter every 3 seconds. The transaction fingerprint in Figure 4(d) is created by a bot that crawls the photo albums of friends and posts spam links at the same time.

E. Transaction Subdivision

Now that we have produced normalized fingerprint matrix for every transaction, we observe that there can be countless types of transactions since every transaction may contain an arbitrary number of actions of various types. Worse, each transaction can be of an arbitrary duration, ranging from a few seconds to a few hours, thus making the feature values of different transactions not comparable against each other and not usable for training. We thus subdivide a transaction further into a limited number of types of primitive **actions**, where actions from transactions of different durations are comparable to each other. For example, an action can be clicking a *Like* button, sending a tweet, or submitting a Facebook comment. It is much easier to differentiate bot actions from real user actions, and once we tell bot actions apart from real user actions, we can separate bot transactions from real user transactions.

We design a new clustering algorithm named **Clustering Based on Density Sort and Valley Point Competition** to subdivide a transaction into actions. We first introduce the following definitions:

1) **Density of a data point:** For a data point p, there is a dataset E that contains all the data points within a radius

of r from p: $E = \{x|dist(x,p) < r\}$. The density of p is then the summation of the values of all the points in E. Using bps values as an example, we then have: $p.density = \sum_{x \in E} x.bps$.

2) **Density of a cluster:** For a cluster C, its density is the density of the point in the cluster that has the highest density.
3) **Potential point of a cluster:** For a cluster C, 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 following cases:
   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 smaller one of the two clusters.

   Here, the two clusters keep separate if the density of the valley point is lower than $\rho\%$ of the density of both clusters. The subdivision only occurs at a valley point whose density is in sharp contrast to the density of surrounding clusters.

Algorithm 1 Clustering algorithm based on density sort and valley point competition

```
Input: dataset D, radius threshold value r
Initialize set C to store clusters
Use r to calculate the density of each data point in D
D := Sort(D)
        ▷ Sort data in D in the decreasing order of density
for 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 the potential point of only one cluster c_a then
        Label e as a member of cluster c_a
    else if e is the potential point of two clusters {c_i, c_{i+1}} then
    ▷ e is a valley point, and e can only be two clusters' valley in two-dimensional space
        competition(e, {c_i, c_{i+1}})
            ▷ Start the valley point competition mechanism
    end if
end for
return C
```

The pseudocode of the algorithm is shown in Algorithm 1. It takes a dataset (D) and a radius threshold value (r) as input. For example, using the 6xN or 4xN matrix from section III-D, D can be a set of data points from a transaction fingerprint 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 and processes the data points in the descending order of density to form and populate 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 then identify the moment where the two clusters meet as a subdivision moment.

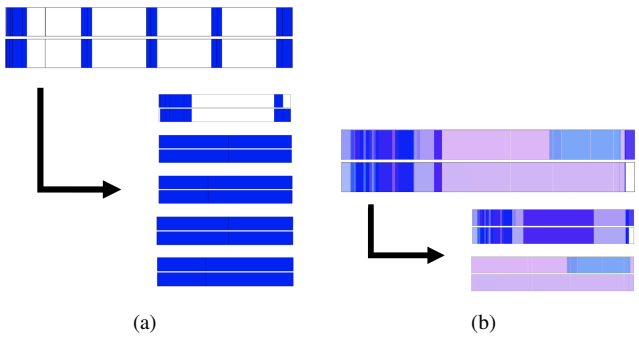


Fig. 5. Transaction subdivision examples

The algorithm then discovers all the subdivision moments of a transaction, which BotFlowMon then uses to divide the transaction into multiple distinct actions. During training, BotFlowMon labels actions from a social bot transaction as *bot actions* and actions from a real user transaction as *real user actions*, respectively. Like a transaction represented by a transaction fingerprint matrix, an action can then be represented by an action fingerprint matrix.

Figure 5 shows two examples of transaction subdivisions. Figure 5(a) shows that by subdividing the transaction fingerprint of a social bot (which is a post bot) into five pieces, every piece now has a more outstanding pattern than the original transaction fingerprint. Figure 5(b), on the other hand, 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, as matched by the subdivision of the transaction into two actions.

F. Machine Learning & Classification

After we use the transaction subvision module (Section III-E) to discover the actions that each transaction contains, we now classify actions into bot actions and real user actions, and then classify the transactions based on how their actions are classified.

We first train an **action classification model** to classify actions. The input to this model is a set of action fingerprints that are labeled as either bot actions or real user actions. To construct the model, we use Keras [29] with TensorFlow [30] to perform the training and classification. Since the training is about nonlinear and high-dimensional data, BotFlowMon uses Multilayer Perceptron (MLP) and Conventional Neural Network (CNN) as its training approaches. We can apply some other machine learning algorithms to the classification module, but BotFlowMon was mainly tested with MLP and CNN.

With the action classification model trained, we then can detect 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. In particular, if all actions are from a real user, we can safely determine the transaction is by a real user; otherwise, we can decide that a transaction is a bot transaction if more than a certain percentage of actions are from a social bot.

IV. Evaluation

A. Data Source

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

We created and labeled the real user and social bot traffic flows as follows, including obtaining the Institutional Review Board (IRB) approval to address ethics, privacy, and human subject issues. For real user traffic flows, we had participants manually conduct normal daily activities on Twitter and Facebook. We adopted an informed consent procedure, which provided the potential participants a clear description about the project before they agree. For social bot traffic flows, we employed a variety of well-used social bot programs, software, and homegrown scripts to conduct bot activities on Twitter and Facebook, we then collected and labeled the corresponding traffic as the ground truth. In order to simulate a variety of social bots and obtain highly credible labeled data, we categorized the social bots into five types according to their implementation mechanisms (section IV-B) and simulated all of them. All of the bots that interacted with Twitter and Facebook were under their terms of service, and their traces were erased after each experiment. We guarantee that the bots we simulated are harmless to campus network, other OSN users, and the OSN environments. For example, we used weather reports, university announcements, and encyclopedic knowledge to simulate fraud and spam posts for post and amplification bots.

We collected 28 gigabytes raw NetFlow data from our experimental platform and 507 gigabytes raw NetFlow data from the university's campus traffic. After the preprocessing, 30,932,991 labeled NetFlow records are involved in the subsequent steps. All the data collected were content-agnostic NetFlow records along with the labeling information. No payload/content data were downloaded and stored during the research process. For the NetFlow records from our campus network, all the internal IP addresses were anonymized, so that we cannot map them back to any individual.

B. Social Bots Simulation

1) Chatbot: Chatbots are very 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 for particular purposes.

The simulation of chatbots relies on some existing widely used chatbot frameworks, APIs, and open-source programs such as botmaster [31], Ontbot [32], and python-twitter API. We created many Twitter and Facebook bot accounts only for research purposes under the OSNs' terms and conditions of service. We collected multitudinous traffic flows of the conversations between real users and these chatbots, with different frequencies of interactions, response times and transmission contents (images, audio files, texts and, hyperlinks).

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 which contain malicious URLs in most cases or malicious texts occasionally [2] [33].

In order to simulate post bot, we downloaded several popular open-source poster bot software from GitHub [34] and wrote some poster bot programs based on APIs such as Tweepy [35] and Facebook API [36]. We ran these bots programs with different frequencies during different time periods to post some harmless messages that contained texts, videos, images and external URLs on Twitter and Facebook.

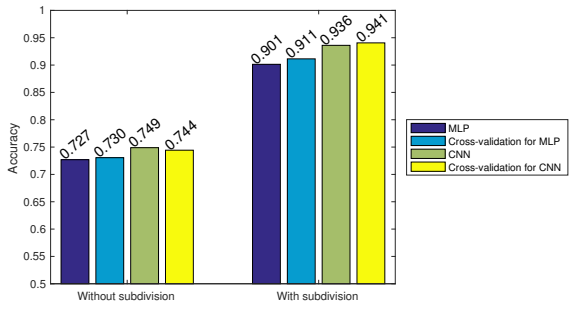
3) Amplification Bot: Amplification bot, due to the large volume of messages it can generate, is often used to create some hot topics for commercial promotion, consensus manipulation, and spam distribution. Without creating new content, amplification bots often work as fake followers, such as those Twitter 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 some unwanted 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. We implemented API-based bot scripts to simulate amplification bots, and used OAuth [37] software for token management and switching accounts.

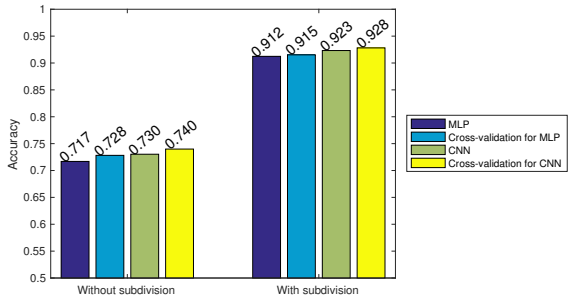
4) OSN Crawler: The core functionality of OSNs is enabling users to share slices of life, personal perspectives, and profiles, however, it can be exploited by crawlers to aggregate data of a large number of OSN users for re-publication or other more nefarious purposes that violate users' privacy and security.

There are two types of OSN crawlers in social networks and we simulated both of them. One is API-based and it relies on a large botnet to dig users' private, sensitive data. Because in OSNs it is common that a user's information can only be seen by their friends, a significant amount of bots are needed to become friends of many users and then access their information. Once the relationship is established, a crawler bot can easily fetch the private data with API functions. The following code exemplifies a crawling process.

```
import twitter
api = twitter.Api()
```



(a) Results for 6x200 transaction fingerprints



(b) Results for 4x200 transaction fingerprints

Fig. 6. Social bot traffic classification accuracy

```
api.GetFollowers()
api.LookupFriendship(user)
```

Another type of OSN crawler is page crawler. Instead of using API privileges, it directly reads the HTML files of OSNs and utilizes 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.

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. Furthermore, it can be a mixture of a social bot and a real user, thus hiding its bot activities behind the normal traffic flows.

C. Results and Analysis

1) Experimental Setup: Based on our empirical studies (of which we skip the details for space considerations), we set the parameters in BotFlowMon as follows:

- For the flow aggregation module, we set the length of every time bin to be 0.1 seconds.
- For the DBSCAN algorithm in the flow aggregation module, we set ϵ to be 10-20 seconds; also, we set $minPts$ to be 1500 bits.
- For the transaction fingerprint generation module, we set N as 200.
- For the transaction subdivision module, we define $\rho\%$ in valley competition mechanism to be 50%.

2) Classification Accuracy: We first look at the accuracy results in classifying social bot traffic. Figure 6(a) shows the test results for the machine learning & classification module with 6x200 transaction fingerprints which use incoming and outgoing bps, pps and ToS features. The test set contains 675 bot transactions and 420 real user transactions. For both Multilayer Perceptron (MLP) and Conventional Neural Network (CNN) algorithms, we used 10-fold cross-validation to check whether the model is overfitting. Since multiple actions had been taken to prevent the model from overfitting, such as controlling the learning rate η and limiting the number of iterations, the cross-validation accuracy is very similar to the testing accuracy and has no sign of overfitting. As we can see in Figure 6(a), the subdivision procedure helps improve the accuracy for about 20%. As a true-or-false classification with the bottom line of 50% accuracy, it is a considerable improvement. For here, CNN achieves the most significant result, with 93.61% of accuracy.

TABLE II
DETAILED RESULTS FOR CNN

	6×200 matrix	4×200 matrix
Accuracy	0.9361	0.9233
Precision	0.9887	0.9821
Recall	0.9067	0.8919
F1 score	0.9459	0.9348

As stated in Section III-D, due to reliability concerns with the ToS features, we also investigated 4x200 transaction fingerprints that do not consider ToS features but use incoming and outgoing bps and pps features in order to train another classification model. The result can be seen in Figure 6(b). Surprisingly, the overall accuracy is only around 1% lower than that of the 6x200 action fingerprints. The decrease of dimensions will reduce the information but also make the model easier to converge, especially for MLP. Again, CNN obtains the best result in this model. Table II shows the detailed evaluation scores for CNN in these two versions of models.

In the real environment, with an accuracy of more than 93%, we can detect most of the bot traffic. Because one bot can create several transaction fingerprints in a specific time window, in order to detect the social bot traffic only one of the transactions need to be identified. A concern here is false alarms as we do not want BotFlowMon to mislabel real user traffic. Currently we label a transaction as a bot transaction if more than 50% of its actions are labeled as bot actions. We found from our experiments that, if a transaction is labeled as a bot transaction only when more than 75% of its actions are classified as bot actions, BotFlowMon will not generate false alarms, but its accuracy will drop down to 89.56%.

3) Subdivision Efficacy: We further evaluated the transaction subdivision module to see whether the Clustering Based on Density Sort and Valley Point Competition algorithm can divide the transaction fingerprints into action fingerprints correctly. Specifically, we studied how different r values in the algorithm could generate different subdivision results and potentially affect the detection outcome. Figure 7 shows the clustering purity scores with different r values. The horizontal

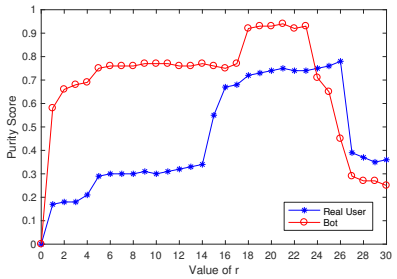


Fig. 7. Purity scores with different r values

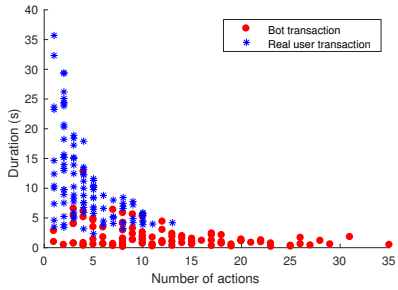


Fig. 8. Scatter Diagram for Subdivision

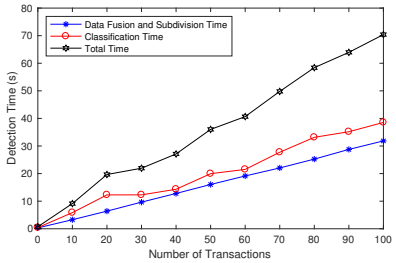


Fig. 9. Social bot detection time

axis represents the values of r and the vertical axis represents the purity scores of the resulted clusters. From Figure 7, we can see that the algorithm is indeed susceptible to the values of the r parameter 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 and can achieve up to 0.9358 purity of the resulting clusters when $r = 23$. 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 contents to real users dynamically, creating NetFlow records that can occupy the gap between actions, imposing challenges and causing impreciseness to the clustering in the algorithm.

However, the ultimate goal of subdivision is not precisely partitioning all the transactions. 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 8 is the scatter plot of the result. While the bot transactions are divided into relatively shorter actions than real user transactions, nonetheless, the durations of bot action fingerprints at 0 to 15 seconds and real user actions in the range of 0 to 40 seconds are comparable with each other, making the subsequent machine learning module easy to converge.

4) BotFlowMon Performance: We tested BotFlowMon on a personal laptop with 2.7-GHz CPU and 16-GB memory to measure the detection performance. The test process only utilized a single thread, and no GPU-based architectures were engaged in computation. The evaluation shows that it takes 0.71 seconds on average to complete the detection procedure for one transaction. The transaction fingerprint generation and subdivision jobs occupy 43.25% of the detection time, while classification process occupies 56.75%. Just as Figure 9 shows, the detection time will increase linearly as the data volume increases. The processing time for flow fingerprint generation and subdivision tends to be more stable than classification. Because different transactions have different amount of action data, the classification workload becomes more uncertain.

Compared with other detection approaches, BotFlowMon has good performance and the ability of real-time monitoring and detection.

V. Limitations and Open Issues

A primary contribution of BotFlowMon is to use data from layers 3 and 4 (i.e., NetFlow data) to detect anomalies at layer 7 (i.e., OSN bot activities). It can accurately identify social bot traffic while protecting OSN users' privacy at the same time. However, BotFlowMon has some limitations:

1) Not all the social bots are malicious, and the boundary between "good" bots and "bad" bots can be blurry. How to distinguish social bots with malicious intentions from those that are innocent is hard to achieve without payload data.
2) Although the training data of BotFlowMon can be enhanced to cover traffic from more socia bots, thus improving the capability of BotFlowMon, nonetheless, if the training does not include traffic from zero-day social bots, the system probably will not be able to detect them.

BotFlowMon also faces several open issues as possible future working items:

1) BotFlowMon currently only uses NetFlow records as its input of traffic flow data. It is worthwhile to extend BotFlowMon to include other traffic formats such as sFlow [26].
2) More features may be explored to improve the accuracy for identifying social bot traffic.
3) BotFlowMon's detection mechanism is at the IP level. When a social bot and a real user interact with an OSN at the same time from the same machine, their traffic will be aggregated into a single transaction, making it hard to distinguish the bot traffic from the real user traffic. We can address this problem as a future work by detecting the social bot traffic from the port level, a finer granularity than the IP level.
4) The current approach is focused on distinguishing the social bot traffic from the real user traffic. Another future work of BotFlowMon can be further identifying different categories of social bots.

VI. 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 tackles big networking data to identify the traffic of OSN bots. As the networking data are layers two and three information and need nothing about OSN content and activities, BotFlowMon departs from previous content-based or OSN-topology-based social bot detection solutions and is a content-agnostic, privacy-preserving and efficient approach.

With no dependence on any OSN content, BotFlowMon devises several new techniques and algorithms, such as an aggregation technique that derives transactions datasets from NetFlow records, a data fusion technique that extracts features from transactions datasets, as well as a density-valley-based clustering algorithm that divides a transaction dataset into multiple actions, altogether enabling a fast and accurate identification of traffic flows from social bots with an accuracy of 92.33–93.61%.

BotFlowMon is also easy to deploy. Any Internet service providers or enterprise networks, as long as they are able to access the traffic flow records such as NetFlow, can deploy BotFlowMon with little need to stretch their bandwidth.

References

[1] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The rise of social bots," *Commun. ACM*, vol. 59, no. 7, pp. 96–104, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2818717

[2] J. Zhang, R. Zhang, Y. Zhang, and G. Yan, "The rise of social botnets: Attacks and countermeasures," *IEEE Transactions on Dependable and Secure Computing*, 2016.

[3] E. Ferrara, "Manipulation and abuse on social media by emilio ferrara with ching-man au yeung as coordinator," *ACM SIGWEB Newsletter*, no. Spring, p. 4, 2015.

[4] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "The socialbot network: when bots socialize for fame and money," in *Proc. of the 27th annual computer security applications conference*, 2011, pp. 93–102.

[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 *Proceedings of the 18th international conference on world wide web*, 2009, pp. 551–560.

[6] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Y. Zhao, "Detecting and characterizing social spam campaigns," in *Proc. of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 35–47.

[7] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social network-based botnet command-and-control: emerging threats and countermeasures," in *International Conference on Applied Cryptography and Network Security*. Springer, 2010, pp. 511–528.

[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 *Proceedings of the 2nd ACM SIGCOMM workshop on Internet measurment*, 2002, pp. 173–174.

[10] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer, "Botornot: A system to evaluate social bots," in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016, pp. 273–274.

[11] J. P. Dickerson, V. Kagan, and V. Subrahmanian, "Using sentiment to detect bots on Twitter: Are humans more opinionated than bots?" in *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2014, pp. 620–627.

[12] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia, "Detecting automation of Twitter accounts: Are you a human, bot, or cyborg?" *IEEE Trans. on Dependable and Secure Computing*, vol. 9, no. 6, pp. 811–824, 2012.

[13] K. Lee, B. D. Eoff, and J. Caverlee, "Seven months with the devils: A long-term study of content polluters on Twitter." in *ICWSM*, 2011.

[14] G. Danezis and P. Mittal, "Sybilinfer: Detecting sybil nodes using social networks." in *NDSS*, 2009, pp. 1–15.

[15] Q. Cao and X. Yang, "Sybilfence: Improving social-graph-based sybil defenses with user negative feedback," *arXiv preprint arXiv:1304.3819*, 2013.

[16] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, "Aiding the detection of fake accounts in large scale social online services," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012, pp. 15–15.

[17] H. Yu, M. Kaminsky, P. B. Gibbons, and A. D. Flaxman, "Sybilguard: defending against sybil attacks via social networks," *IEEE/ACM Transactions on networking*, vol. 16, no. 3, pp. 576–589, 2008.

[18] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *IEEE Symposium on Security and Privacy*, 2008, pp. 3–17.

[19] 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.

[20] 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.

[21] J. Ratkiewicz, M. Conover, M. R. Meiss, B. Gonçalves, A. Flammini, and F. Menczer, "Detecting and tracking political abuse in social media." *ICWSM*, vol. 11, pp. 297–304, 2011.

[22] G. Wang, M. Mohanlal, C. Wilson, X. Wang, M. Metzger, H. Zheng, and B. Y. Zhao, "Social turing tests: Crowdsourcing sybil detection," *arXiv preprint arXiv:1205.3856*, 2012.

[23] 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*, vol. 5, no. 2, 2008, pp. 139–154.

[24] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "Botfinder: Finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th international conference on emerging networking experiments and technologies*. ACM, 2012, pp. 349–360.

[25] 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*. IEEE, 2007, pp. 715–720.

[26] 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.

[27] C. Orsini, A. King, D. Giordano, V. Giotsas, and A. Dainotti, "BGPStream: a software framework for live and historical BGP data analysis," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 429–444.

[28] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, no. 34, 1996, pp. 226–231.

[29] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[30] "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[31] 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.

[32] H. Al-Zubaide and A. A. Issa, "Ontbot: Ontology based chatbot," in *2011 Fourth International Symposium on Innovation in Information & Communication Technology (ISIICT)*, 2011, pp. 7–12.

[33] C. Grier, K. Thomas, V. Paxson, and M. Zhang, "@spam: the underground on 140 characters or less," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 27–37.

[34] anonymity, "Poster bots from github," https://github.com/search?q=poster+bot, 2018.

[35] J. Roesslein, "tweepy documentation," *Online] http://tweepy. readthedocs. io/en/v3*, vol. 5, 2009.

[36] W. Graham, *Facebook API developers guide*. Infobase Publishing, 2008.

[37] D. Hardt, "The OAuth 2.0 authorization framework," 2012.