Clickstream User Behavior Models

GANG WANG, Virginia Tech and UC Santa Barbara
XINYI ZHANG and SHILIANG TANG, UC Santa Barbara
CHRISTO WILSON, Northeastern University
HAITAO ZHENG and BEN Y. ZHAO, UC Santa Barbara

The next generation of Internet services is driven by users and user-generated content. The complex nature of user behavior makes it highly challenging to manage and secure online services. On one hand, service providers cannot effectively prevent attackers from creating large numbers of fake identities to disseminate unwanted content (e.g., spam). On the other hand, abusive behavior from real users also poses significant threats (e.g., cyberbullying).

In this article, we propose clickstream models to characterize user behavior in large online services. By analyzing clickstream traces (i.e., sequences of click events from users), we seek to achieve two goals: (1) detection: to capture distinct user groups for the detection of malicious accounts, and (2) understanding: to extract semantic information from user groups to understand the captured behavior. To achieve these goals, we build two related systems. The *first* one is a *semisupervised* system to detect malicious user accounts (Sybils). The core idea is to build a clickstream similarity graph where each node is a user and an edge captures the similarity of two users' clickstreams. Based on this graph, we propose a coloring scheme to identify groups of malicious accounts without relying on a large labeled dataset. We validate the system using ground-truth clickstream traces of 16,000 real and Sybil users from Renren, a large Chinese social network. The *second* system is an *unsupervised* system that aims to capture and understand the fine-grained user behavior. Instead of binary classification (malicious or benign), this model identifies the natural groups of user behavior and automatically extracts features to interpret their semantic meanings. Applying this system to Renren and another online social network, Whisper (100K users), we help service providers identify unexpected user behaviors and even predict users' future actions. Both systems received positive feedback from our industrial collaborators including Renren, LinkedIn, and Whisper after testing on their internal clickstream data.

CCS Concepts: • **Security and privacy** → **Social network security and privacy**; • **Human-centered computing** → *Empirical studies in HCI*; • **Computing methodologies** → *Cluster analysis;*

Additional Key Words and Phrases: Clickstream, behavior model, online social networks, spam and abuse

ACM Reference Format:
Gang Wang, Xinyi Zhang, Shiliang Tang, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. 2017. Clickstream user behavior models. ACM Trans. Web 11, 4, Article 21 (July 2017), 37 pages.
DOI: http://dx.doi.org/10.1145/3068332

1. INTRODUCTION

The next generation of Internet services is driven by users and user-generated content. Whether it's online social networks, online review services (Yelp, TripAdvisor), content

This work is supported by the National Science Foundation, under grants CNS-1527939 and IIS-1321083.
Authors' addresses: G. Wang, Department of Computer Science, Virginia Tech, 2202 Kraft Drive, Knowledge Works II, Blacksburg, VA, 24060; email: gangwang@vt.edu; X. Zhang, S. Tang, H. Zheng, and B. Y. Zhao, Department of Computer Science, UC Santa Barbara, 2104 Harold Frank Hall, Santa Barbara, CA 93106; emails: {xyzhang, shiliang_tang, htzheng, ravenben}@cs.ucsb.edu; C. Wilson, College of Computer and Information Science, Northeastern University, 440 Huntington Ave 202, Boston, MA 02115; email: cbw@ccs.neu.edu.

DOI: http://dx.doi.org/10.1145/3068332

sharing communities (Reddit), or crowdsourcing systems (Amazon Turk, TaskRabbit), their success is increasingly dependent on active and well-behaved users.

Due to the complex and diverse nature of user behavior, it is more challenging than ever to manage and secure these online services. On one hand, service providers cannot prevent attackers from creating large numbers of fake identities (Sybils) to disseminate unwanted content [15]. Existing defenses such as online Turing tests (CAPTCHAs) are routinely solved by dedicated workers for pennies [39], and even complex tasks can be overcome by a growing community of malicious crowdsourcing services [41, 61]. The result is a dramatic rise in forged and malicious content such as fake reviews on Yelp [56], malware and spam on social networks [5, 18, 20, 29], and rogue political lobbying campaigns [52]. On the other hand, even among the the real-user population, toxic and abusive behavior such as cyberbullying [7, 26] and trolling [33] is significantly threatening the well-being of online communities.

Recent work has explored a number of solutions to detect fake accounts (Sybils) and abusive behavior. Existing Sybil detection systems mostly rely on social graphs [14, 55, 68, 69], with a key assumption that Sybils have difficulty befriending real users. This forces Sybils to connect to each other to form strongly connected subgraphs and makes them detectable for community detection algorithms [57]. However, recent measurements show that real-world Sybils have broken this assumption by integrating themselves into real user communities [66].

In an effort to detect abusive behavior, existing approaches often rely on *supervised* machine learning to build classifiers based on predefined features [3, 58, 67]. However, these systems require large ground-truth data for training. Once the models are trained, they only detect the same malicious behavior in the training data and fail to identify new attacks. More importantly, machine-learning models often serve as a "blackbox," which offers little knowledge about how attackers (as well as real users) behave, and how their behavior changes over time.

In this article, we describe a new approach toward *detecting* and *understanding* user behavior in online systems for combatting abuse. The core idea is to build detailed user behavior models using *clickstream* traces. Clickstreams are timestamped server-side traces of click events, generated by users during their web browsing "sessions" or interactions with mobile apps [21, 36, 45]. Our goal is to build clickstream models to detect previously unknown attacks (e.g., Sybil accounts) without relying on large ground-truth data. More importantly, we seek to provide mechanisms to interpret and understand captured behavior and track behavior changes over time.

Our efforts lead to two related clickstream analytics systems for the *detection* and *interpretation* of user behavior. The first system takes a semisupervised approach to detect malicious user accounts (Sybils). The second system is unsupervised, extending the basic model to interpret fine-grained user (attacker) behavior.

First, we build a Sybil detection system based on the intuition that Sybils and real users are likely to have different click patterns when they interact with online services. For example, real users likely partake of numerous actions, while Sybils focus on specific actions (i.e., acquiring friends and disseminating spam) to maximize utility per time spent. These differences will manifest as significantly different (and distinctive) patterns in clickstreams, making them effective tools for "profiling" user behavior. To capture the different "profiles" of user behaviors, we build a clickstream similarity graph to capture pairwise "similarity distance" between user clickstreams and apply clustering to identify groups of user behavior patterns. We validate our models using ground-truth clickstream traces from 16,000 real and Sybil users from Renren, a large Chinese social network. Our system only requires a small set of known "good" users to color captured behavior clusters, eliminating the need for large ground-truth data.

We apply our system to the real social networks Renren and LinkedIn and detected previously unknown attacks.

To further provide semantic interpretations on captured behavior, we build a second system as an extension of the previous clickstream model. Instead of simply performing binary classification on users, this model identifies natural clusters of users and extracts key features to interpret captured behavior. More specifically, we propose an *iterative feature pruning* algorithm to partition the clickstream similarity graph, which removes the influence of dominating features from each subsequent layer of clusters. The result is a hierarchy of clusters, where higher-level clusters represent more general user behavior patterns and lower-level clusters further identify smaller groups that differ in key behavioral patterns. We can further use Chi-square statistics to identify statistical features that can be used to categorize and label behavior groups. We validate the system using the Renren dataset mentioned earlier, and another 135 million click events from 100K users on Whisper, a popular anonymous social network app. Our system revealed key insights about users on both networks, including identifying and predicting dormant users (Whisper), capturing hostile behaviors during private chat (Whisper), and revealing different Sybil attack strategies that seek to evade existing defenses (Renren).

Our article makes four key contributions toward detecting and understanding user/attacker behavior in online services.

- We propose a novel *semisupervised* system to detect malicious users. Our system is based on a clickstream similarity graph to capture user clusters to differentiate normal and malicious (Sybil) users. Our Sybil detector requires minimal input from the service providers (semisupervised). Experiments using real-world clickstream data show accurate detection results (<1% false positives and <4% false negatives).
- We extend the basic model for a new *unsupervised* system to interpret the captured user clusters. With an *iterative feature pruning* algorithm, we capture user behavior models as hierarchical clusters. Our tool automatically produces key features to interpret the meaning of the clusters. Applying this system to two real-world clickstream datasets (142 million clicks in total) helps to identify unexpected user behavior (malicious accounts in Renren, hostile chatters in Whisper), and even predict users' future actions.
- We build a visualization interface to visualize the clustering results.[1] Our user study shows this tool is easy to use. People can effectively extract semantic labels for given behavior clusters and interpret the meaning of the captured behavior.
- Working closely with industrial collaborators, we have deployed our Sybil detector in real-world social networks (Renren and LinkedIn) and captured previously unknown attacks. Our unsupervised behavior model and the visualization tool have received positive feedback from the Whisper Data Science team.

2. RELATED WORK

User Behavior Modeling in Online Services. Understanding user behavior is important to the design and operation of online services. Recent works analyze network traffic and server logs to study users' search intent [43] and Wikipedia editing patterns [19] and to characterize user activities in online social networks [4, 47].

Clickstream Analysis. Earlier research used clickstream data for Web Usage Mining [2, 23, 24, 48]. Researchers applied simple methods such as Markov Chains to capture users' navigation paths within a website [4, 25, 36, 45]. However, these models focus on the simple aspects of user behavior (e.g., users' favorite webpage) and are incapable

[1] Our tool is available for sharing: http://sandlab.cs.ucsb.edu/clickstream.

of modeling more sophisticated user behavior. Other approaches use clustering techniques to identify user groups that share similar clickstream activities [21, 50, 54]. The resulting clusters can be used to infer user interests [50] or predict future user behaviors [21]. However, existing clustering-based models are largely supervised, requiring large samples of ground-truth data to train or fine-tune parameters [45, 54]. Also, many behavioral models are built as "black boxes" for classification tasks, offering little explanations on how users behave and why [21]. Our work seeks to build unsupervised (or semisupervised) clickstream behavioral models and produce intuitive explanations on the models.

Sybil Attack and Defenses. One of the primary goals of analyzing user behavior is to detect malicious users and activities. Sybils (i.e., groups of fake identifies) are the foundation of many online attacks such as spam, scam and political campaigns, malware distribution, and identity theft (collecting personally identifiable information) [18, 20, 52]. The major body of existing work leverages social graphs to detect Sybils. These systems detect tight-knit Sybil communities that have a small quotient cut from the honest region of the graph [10, 14, 57, 68, 69]. However, recent studies have demonstrated the limitations of this approach. Yang et al. show that Sybils on Renren blend into the social graph rather than forming tight communities [66]. Mohaisen et al. show that many social graphs are not fast-mixing, which is a necessary precondition for community-based Sybil detectors to be effective [38].

A second body of work has used supervised classifiers to detect Sybil behavior on Twitter [3, 67], Facebook [49], and Amazon [42]. However, relying on specific features (e.g., URLs, content, followers) makes these systems vulnerable to Sybils with different attack strategies. All of this motivates us to develop novel user behavior models to detect Sybils. In this work, we build Sybil detectors based on clickstream models (semisupervised), which do not rely on specific assumptions on Sybil behaviors and are resilient to their behavior changes.

Finally, recent works propose to detect spammers and Sybils based on synchronized behaviors, that is, performing similar tasks around the same time [5, 6, 11, 29, 30, 35]. These systems build graphs based on users and their actions such as Facebook likes [6], Twitter following [30], and YouTube comments [35]. Then they use semisupervised clustering to detect suspicious user groups with synchronized actions. These systems along with ours demonstrate the value of semisupervised learning in detecting new attacks. Our system differs from existing ones: instead of focusing on a specific action (liking or following), we use clickstreams to capture a wider range of user actions to model their behavior.

Clickstream Visualization. Researchers have developed interactive interfaces to visualize and inspect clickstream data. Existing tools generally focus on visualizing raw user clicks [37], click event sequences [70], or click transitions [63]. Instead, we build a tool to visualize clickstream behavioral clusters and provide hints for understanding user behavior patterns.

3. CLICKSTREAM DATA AND PRELIMINARY ANALYSIS

In this work, we seek to apply clickstream analysis to detect malicious attackers (i.e., Sybils) and interpret complex user behavior. To provide context, we first describe the clickstream datasets used in our study. We obtained server-side clickstream data from two large-scale online social networks: Renren and Whisper. The Renren dataset contains ground-truth normal user accounts and Sybil accounts and will be used to evaluate both proposed systems. The Whisper dataset only contains normal users and will be exclusively used to evaluate the second system for behavior interpretation. In the

Table I. Clickstream Datasets from Whisper and Renren

Dataset	Time	# of Users	# of Events
Whisper	Oct. 13–Nov. 26, 2014	99,990	135,208,159
Renren-Normal	Mar. 31–Apr. 30, 2011	5,998	5,856,941
Renren-Sybil	Feb. 28–Apr. 30, 2011	9,994	1,008,031

following, we briefly introduce the two datasets and perform a preliminary analysis to highlight the challenges in detecting malicious users and understanding user behavior.

3.1. Renren

Renren is one of the largest online social networks in China, with 236 million users as of 2016.[2] Renren offers similar functionalities as Facebook, allowing users to maintain a personal profile and build social connections. Users can post status updates, write blog entries, share photos, and interact with other users' content (e.g., liking and commenting). The key difference between Renren and Facebook is Renren's "footprint" feature, which allows users to see who has recently visited their profiles.

We obtained detailed clickstream data from Renren in collaboration with Renren's Security team. A clickstream is the sequence of HTTP requests made by a user to the Renren website.[3] Most requests correspond to a user explicitly fetching data by clicking a link or a button, although some requests may be programmatically generated. A clickstream can be unambiguously attributed to a specific user account, for example, by examining the HTTP request cookies.

Our Renren dataset contains 5,998 normal users and their clickstream traces over 2 months in 2011 (Table I). These users were selected uniformly at random from the Renren user population and were manually verified by Renren's security team. In addition, the dataset includes 9,994 Sybil accounts randomly sampled from all previously banned accounts by Renren. These accounts were flagged by a number of internal detectors (e.g., URL blacklists, spam filters) and crowdsourced user reports. Sybil accounts in our dataset were manually verified by a volunteer team of Renren. Note that Renren allows misbanned users to claim their accounts back by calling Renren's customer support. Sybil accounts in our dataset have been banned for at least a year without being claimed, further eliminating potential false positives. Regarding false negatives, there should be Sybil accounts that have not been flagged by existing detectors. We will use this dataset to develop and test new Sybil detection systems to capture previously unknown Sybils.

In this dataset, each click event is characterized by userID, timestamp, event type, and event parameter. The userID in our dataset (including Whisper data) is globally unique and has been fully anonymized to protect user privacy. We obtained userIDs from each company through internal collaborators. Our study has been approved by our local IRB under protocol #COMS-ZH-YA-010-6N. There are 55 types of events grouped into eight primary categories. These categories cover the major user actions on Renren but exclude administrative events such as changing profiles and resetting passwords:

- **Friending:** Sending friend requests, accepting or denying those requests, and unfriending.
- **Photo:** Uploading photos, organizing albums, tagging friends, browsing photos, and writing comments.
- **Profile:** Browsing user profiles. Profiles on Renren can be browsed by anyone, but the displayed information is restricted by the owner's privacy settings.

[2] http://www.renren-inc.com/en/.

[3] Renren also has a mobile version, but it is not as popular as the website at the time of our experiments.

Table II. Event Types in the Renren Dataset

		Sybil Clicks		Normal Clicks	
Category	**Description**	**# (K)**	**%**	**# (K)**	**%**
Friending	Send request	417	**41**	16	0
	Accept invitation	20	2	13	0
	Invite from guide	16	2	0	0
Photo	Visit photo	242	**24**	4,432	**76**
	Visit album	25	2	330	**6**
Profile	Visit profiles	160	**16**	214	4
Share	Share content	27	3	258	**4**
Message	Send IM	20	2	99	2
Blog	Visit/reply blog	12	1	103	2
Notification	Check notification	8	1	136	2

# of click events are presented in thousands. Activities with <1% of clicks are omitted for brevity. All of the events are user-initiated events.

- **Sharing:** Users posting URLs linking to videos, blogs, or photos in/outside Renren.
- **Message:** Status updates and instant messages.
- **Blog:** Reading/writing blogs and commenting.
- **Notification:** Users actively clicking on the notifications.
- **Like:** Users liking (or unliking) content.

Table II displays the most popular events. Note that the percentages for click events are calculated for Sybils and normal users separately; that is, each "%" column sums to 100%. We find Sybils and normal users behave differently. Normal users spend most of their clicks on viewing photos (76%), albums (6%), and sharing (4%). In contrast, Sybils are skewed to making friend requests (41%), viewing photos (24%), and browsing profiles (16%). On Renren, it makes sense for malicious accounts (Sybils) to try to become friends with normal users, because only by doing so can they access the user's private information and share content (spam) with the user. In addition, passively viewing photos may correspond to data scrapers (crawlers) that massively collect profile photos and sell to third parties (e.g., dating websites). However, given that other attacks are possible (e.g., manipulating trending topics [27], passively collecting friends [53]), we cannot rely on these features alone to identify Sybils.

Note that normal users and Sybils share content (4% and 3%, respectively) as well as send messages (2% and 2%, respectively) at similar rates. It is likely that spammers try to be less aggressive to avoid detection. This is an important observation, because sharing and messaging are the primary channels for spam dissemination on Renren. The similar rates of legitimate and illegitimate sharing/messaging indicate that spam detection systems cannot simply leverage numeric thresholds to detect spam content.

3.2. Whisper

Whisper is a popular smartphone app for anonymous social messaging. It allows users to share confessions and secrets under anonymous nicknames without worrying about privacy [13, 60]. As of December 2015, Whisper has reached 20 million users.[4] Unlike traditional social networks, Whisper does not maintain user profiles or social connections. Its key function is messaging: the app overlays a user's short text message on top of a background picture selected by keywords. For example, a message that describes a "funny moment" may be automatically attached to a background picture of a smiling face. The resulting *whisper* message, looking like an Internet meme, is posted to the public stream, where other users can read, reply, or heart (like) it. In addition, the app

[4] http://money.cnn.com/2015/12/11/technology/whisper-20-million-users-privacy.

Table III. Event Types in the Whisper Dataset

Category	Event Type	Events # (K)	Events %	Initiated by User?
Browsing	View whisper	52,437	38	Yes
	View popular feed	16,008	12	Yes
	View nearby feed	5,354	4	Yes
	View latest feed	2,346	2	Yes
	View other feed	196	1	Yes
Account	Login	16,994	12	Yes
Posting	Heart whisper	2,156	2	Yes
	Upload image	1,325	1	Yes
	Create whisper	1,308	1	Yes
Chatting	Being blocked in chat	3,271	3	No
	Block user in chat	3,271	3	Yes
	Start a chat	2,238	2	Yes
Notification	Receive notification	9,680	7	No
	Whisper recommendation	2,530	2	No

# of click events are presented in thousands. Events that are <1% are omitted for brevity.

provides a chat feature to facilitate direct communication. Any user can start a private chat with the whisper author. Finally, users browse whispers from several public lists.

We collect detailed clickstream data from Whisper in collaboration with Whisper's Data Science team. The dataset contains 135 million click events from 99,990 users over 45 days in 2014 (Table I). Users were randomly selected from the Whisper user populations that have at least 250 clicks (Figure 1) over this time period as a representative sample. Our Whisper dataset does not contain Sybil accounts. To the best of our knowledge, Sybils are not a primary threat to the Whisper network. The Whisper dataset contains 33 types of events grouped into six categories. These categories are:

- **Browsing:** Browsing whispers, visiting the public whisper feeds (popular/nearby/latest list).
- **Account:** Creating a user account and login in the app.
- **Posting:** Posting original whispers and replies, hearting/unhearting a whisper, sharing whispers, and tagging a whisper to a topic.
- **Chatting:** Initiating a chat, blocking other users in a chat, and being blocked in a chat.
- **Notification:** Receiving notifications about hearts/replies on their whispers, and whisper recommendations.
- **Spam:** Whispers being examined or deleted by system admins, flagging other people's whispers. Events in this category are all below 1% (omitted from Table III).

Among the 33 event types, 25 are user-initiated events corresponding to the user performing an action on the app (e.g., "posting a whisper"). The other eight events are system events that don't require user action (e.g., "receiving notifications"). Table III shows the most popular events and the absolute number (in thousands) and the percent of clicks. Overall, the most prevalent events are related to content consumption such as viewing whispers. Interestingly, under the chatting category, the most prevalent events are "blocking users" and "being blocked" by others. Intuitively, an anonymous environment is more likely to foster abusive behaviors (e.g., bullying) [51]. Later, we investigate this behavior in greater detail using behavioral models.

Our Whisper dataset also contains the content of the public whispers (about 1 million) posted by these users. This content data is not used to construct clickstreams, but to

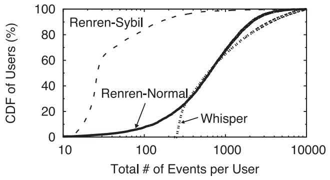


Fig. 1. Number of click events per user.

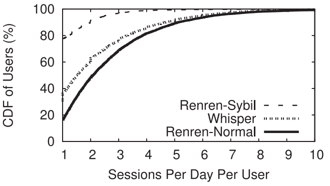


Fig. 2. Number of sessions per day per user.

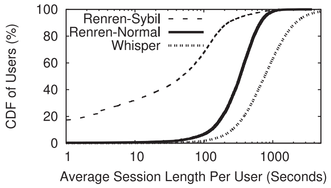


Fig. 3. Average session length per user.

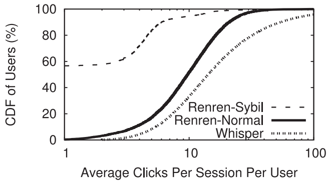


Fig. 4. Average number of click events per session per user.

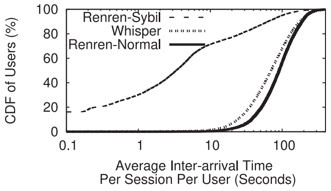


Fig. 5. Average click time interval per session per user.

understand specific user behavior and user intent later in our analysis. We don't have any content data in the Renren dataset.

3.3. Session-Level Characteristics

To provide contexts for user activities in Renren and Whisper, we analyze their *session-level* characteristics. Each user's clickstream can be divided into sessions, where a session represents the sequence of a user's clicks during a single visit to Renren (or Whisper). Unfortunately, users do not always explicitly end their session by logging out. As in prior work, we assume that a user's session is over if they do not make any requests for 20 minutes [4]. Session duration is calculated as the time interval between the first and last click within a session. Overall, Renren traces contain 113,595 sessions for Sybils and 467,179 sessions for normal users. Whisper traces contain 2,440,264 sessions in total. In the following, we examine their session-level statistics as well as the click transitions within sessions.

Session Length and Frequency. Figures 2 to 5 show different statistics regarding the sessions of Renren normal users and Sybils, and Whisper users. We find that both Renren normal users and Whisper users behave significantly different from Sybil accounts. More specifically, Sybils have fewer sessions per day. As shown in Figure 2, 80% of Sybil accounts only have one session per day. In addition, the duration of Sybil sessions is also much shorter with fewer clicks (Figures 3 and 4): 70% of Sybil sessions are <100 seconds long, while the vast majority of normal sessions last several minutes. Finally, as shown in Figure 5, the average interarrival time between Sybil clicks is an order of magnitude shorter than for normal clicks. This indicates that Sybils do not linger on pages, and some of their activities may be automated.

The observed session-level Sybil characteristics are driven by attackers attempts to circumvent Renren's security features. Renren limits the number of actions each account can take, (e.g., 50 friend requests per day and 100 profiles browsed per hour). Thus, in order to maximize efficiency, attackers create many Sybils, quickly log in to each one, and perform malicious activities (e.g., sending unsolicited friend requests and spam), then log out and move to the next Sybil. As shown in Table II, Sybils

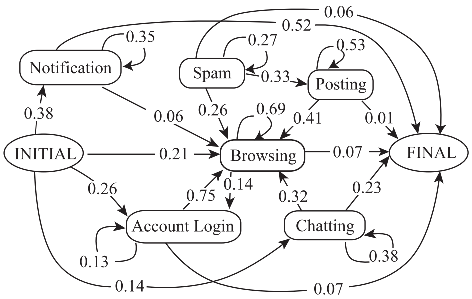


Fig. 6. State transitions for Whisper users.

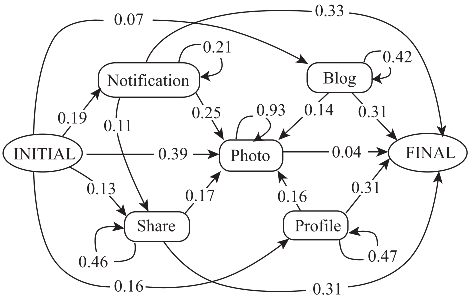


Fig. 7. State transitions for Renren normal users.

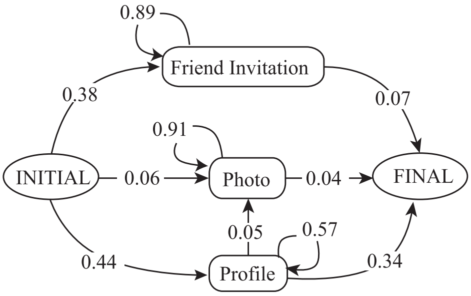


Fig. 8. State transitions for Renren Sybils.

spend a great deal of clicks sending friend requests and browsing profiles, despite Renren's security restrictions.

In addition, we observe that Whisper users have longer sessions than Renren users (Figure 3 and Figure 4). One possible explanation is that Whisper takes advantage of the always-on smartphones, which enables easier access to the service (e.g., users can log in to the service even when they are walking or taking the bus).

Click Transitions. Next, we examine differences in their click ordering; that is, how likely is it for a user to transition from activity A to activity B during a single session? We use a Markov Chain model to analyze click transitions. In this model, each state is a click category, and edges represent transitions between categories. We add two abstract states, initial and final, that mark the beginning and end of each click session. Figure 6 shows the category transition probabilities for Whisper users, and Figure 7 and Figure 8 are the results for normal users and Sybils in Renren. The sum of all outgoing transitions from each category is 1.0. To reduce the complexity of the diagram, edges with probability $<5\%$ are pruned (except for transitions to the final state).

Figure 8 demonstrates that Sybils follow a very regimented set of behaviors. After logging in, Sybils immediately begin with friend invitation spamming or profile browsing. The profile browsing path represents crawling behavior: the Sybil repeatedly views user profiles until their daily allotment of views is exhausted. Sybils don't often start with photo crawling (0.06). But once they get started, they are likely to keep crawling photos without switching to other actions (0.91).

On the contrary, Figures 6 and 7 show that normal users engage in a wider range of activities, and the transitions between states are more diverse. For Renren normal users, the highest centrality category is photos, and it is also the most probable state after login. Intuitively, users start from their newsfeed, where they are likely to see and click on friends' recent photos. The second most probable state after login is checking recent notifications. Similar for Whisper, the most probable state after login is checking notifications and browsing whisper messages. Note that some people will go through "login" again[5] before browsing, if they have closed the app on the phone. For both Renren and Whisper users, actions related to generating new content (e.g., Sharing, Messaging, Posting) have low probability states. This is consistent with the results of earlier studies, which show that content generation is less frequent than consumption in online social networks [28, 64].

3.4. Discussion

In summary, we have analyzed the clickstream data from Renren and Whisper to characterize user behavior from three angles: click events, sessions, and click transitions. We have two key observations from our initial data analysis:

Sybil Detection. Our analysis reveals the different behaviors of Sybil and normal user accounts on Renren, which suggests the possibility of using clickstream data for Sybil detection. Here we first explore a supervised Sybil detection method by directly applying these features, and discuss why this approach is limited in practice.

For this experiment, we extract features from session-level information and click activities, and build a Support Vector Machine (SVM) [44] classifier. These features include four session features (average clicks per session, average session length, average interarrival time between clicks, and average sessions per day) and eight activity features (percentage of clicks in each of the eight Renren event categories). We computed values for all 12 features for all users in the Renren dataset, input the data to an SVM, and ran 10-fold cross-validation. The resulting classification accuracy was 98.9%, with 0.8% false positives (i.e., classify normal users as Sybils) and 0.13% false negatives (i.e., classify Sybils as normal users).

While our SVM results are quite good, a supervised approach has key limitations. In practice, we would like to avoid detection models that are heavily dependent on large ground-truth datasets, since the resulting detector can only capture attackers observed in the past (biased). In addition, Sybils can apply specific changes to their behavior to evade the detection [66]. For a practical Sybil detection system, we seek to develop clickstream analysis techniques that leverage unsupervised learning on real-time data samples, i.e., require zero or little ground truth. In the following, we will focus on developing clickstream models for real-time, unsupervised Sybil detection (Sections 4, 5, and 6) and detecting new attacks in real-world online social networks (Section 7).

From Sybil Detection to Behavior Modeling and Interpretation. Sybil detection only treats users in a binary fashion, while user behaviors in today's online services are more complex. Tools like session analysis or Markov Chain models can only scratch the surface. First, these tools treat all the users as a single population (or group) and analyze their behaviors as a whole. This is insufficient to identify the fine-grained, different user types or behaviors in the service. For example, even within the Sybil accounts, there are likely different attacking strategies by different attackers, which are hidden from the current analysis. Second, as shown in Figures 6 to 8, user behavior can be very complex even though we only look at the high-level "categories" of their click events. To these ends, we will further build a generic clickstream analysis tool to identify the natural groups of user behaviors. More importantly, we provide mechanisms to help service providers to effectively interpret and understand the captured behavior (Sections 8, 9, 10, and 11).

[5] Login is usually automated, without requiring users to enter a password.

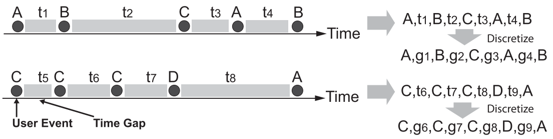


Fig. 9. Discretizing two clickstreams into event sequences.

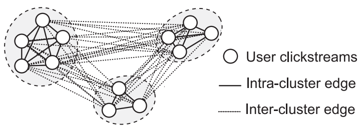


Fig. 10. Clickstream similarity graph.

4. CLICKSTREAM MODELING AND CLUSTERING

We start by addressing the “detection” part of behavior modeling (Section 4–7) and will discuss the “interpretation” aspect in later sections (Section 8–11). In the following, we build a clickstream model to detect Sybils in online social networks. As shown in earlier analysis, clickstream data for Sybils and normal users captured the differences in their behavior. We build models of user activity patterns that can effectively distinguish Sybils from normal users. Our goal is to cluster similar clickstreams together to form general user “profiles” that capture specific activity patterns. We then leverage these clusters (or profiles) to build a Sybil detection system. The current section (Section 4) focuses on the clickstream model, and in Section 5, we develop an incremental Sybil detector that can scale with today’s large social networks. We extend this detector in Section 6 by proposing a semisupervised Sybil detector, where only a minimal (and fixed) amount of ground truth is needed. Finally, in Section 7, we describe the experimental experience of testing our prototype code in real-world social networks (Renren and LinkedIn).

We begin this section by defining three models to represent a user’s clickstream. For each model, we describe similarity metrics that allow us to cluster similar clickstreams together. Finally, we use our ground-truth data to evaluate the efficacy of each model in distinguishing Sybils from normal users. We build upon these results later to develop practical Sybil detection systems based on clickstream analysis.

4.1. Clickstream Models

We start by defining clickstream models. At the high level, we construct one clickstream for each user to capture all his or her clicks, since our Sybil detection is at the user account level. We did not divide the clickstream into sessions, because individual sessions can be too short to model a user’s behavior. Next, we present three models to capture a user’s clickstream.

Click Sequence Model. We start with the most straightforward model, which only considers click events. As shown in Section 3, Sybils and normal users exhibit different click transition patterns and focus their energy on different activities. The Click Sequence (CS) model treats each user’s clickstream as a sequence of click events, sorted by order of arrival.

Time-Based Model. As shown in Figure 5, Sybils and normal users generate click events at different speeds. The Time-based model focuses on the distribution of gaps between events: each user’s clickstream is represented by a list of interarrival times $[t_1, t_2, t_3, \ldots, t_n]$, where n is the number of clicks in a user’s clickstream.

Hybrid Model. The Hybrid model combines click types and click interarrival times. Each user’s clickstream is represented as an in-order sequence of clicks along with interevent gaps between clicks. An example is shown in Figure 9: $[A, t_1, B, t_2, C, t_3, A, t_4, B]$, where A, B, C are click types, and t_i is the time interval between the i^{th} and $(i+1)^{th}$ event.

Click types. Both the Click Sequence model and the Hybrid model represent each event in the sequence by its click event type. We note that we can control how granular the event types are in our sequence representation. One approach is to encode clicks based on their specific *activity*. Renren's logs define 55 unique activities. Another option is to encode click events using their broader *category*. In our dataset, our 55 activities fall under eight click categories (see Section 3). Our experimental analysis evaluates both representations to understand the impact of granularity on model accuracy.

4.2. Computing Sequence Similarity

Having defined three models of clickstream sequences, we now move on to investigating methods to quantify the similarity between clickstreams. In other words, we want to compute the *distance* between pairs of clickstreams. First, we discuss general approaches to computing the distance between sequences. Then we discuss how to apply each approach to our three clickstream models.

4.2.1. Defining Distance Functions. We define three distance functions as follows:

Common Subsequences. The first distance metric involves locating the common subsequences of varying lengths between two clickstreams. We formalize a clickstream as a sequence $S = (s_1 s_2 \ldots s_i \ldots s_n)$, where s_i is the i^{th} element in the sequence. We then define T_N as the set of all possible k-grams (k consecutive elements) in sequence S where $k \leq N$: $T_N(S) = \{k\text{-}gram | k\text{-}gram = (s_i s_{i+1} \ldots s_{i+k-1}), i \in [1, n+1-k], k \in [1, N]\}$. Simply put, each k-gram in $T_N(S)$ is a subsequence of S with a length of k. Finally, the distance between two sequences can then be computed based on the number of common subsequences shared by the two sequences. Inspired by the *Jaccard Coefficient* [34], we define the distance between sequences S_1 and S_2 as

$$D_N(S_1, S_2) = 1 - \frac{|T_N(S_1) \cap T_N(S_2)|}{|T_N(S_1) \cup T_N(S_2)|}. \tag{1}$$

We will discuss the choice of N in Section 4.2.2.

Common Subsequences with Counts. The common subsequence metric defined previously only measures distinct common subsequences; that is, it does not consider the frequency of common subsequences. We propose a second distance metric that rectifies this by taking the *count* of common subsequences into consideration. For sequences S_1, S_2 and a chosen N, we first compute the set of all possible subsequences from both sequences as $T = T_N(S_1) \cup T_N(S_2)$. Next, we count the frequency of each subsequence within each sequence i ($i = 1, 2$) as array $[c_{i1}, c_{i2}, \ldots, c_{in}]$, where $n = |T|$. Finally, the distance between S_1 and S_2 can be computed as the *Euclidean Distance* between the two arrays:

$$D(S_1, S_2) = \frac{1}{\sqrt{2}} \sqrt{\sum_{j=1}^{n} (c_{1j} - c_{2j})^2}. \tag{2}$$

Distribution-Based Method. Unfortunately, the prior metrics cannot be applied to sequences of continuous values (i.e., the Time-based model). Instead, for continuous-value sequences S_1 and S_2, we compute the distance by comparing their value distribution using a two-sample Kolmogorov-Smirnov test (*K-S* test). A two-sample *K-S* test is a general nonparametric method for comparing two empirical samples. It is sensitive to differences in location and shape of the empirical Cumulative Distribution Functions (CDFs) of the two samples. We define the distance function using K-S statistics:

$$D(S_1, S_2) = sup_t |F_{n,1}(t) - F_{n',2}(t)|, \tag{3}$$

where $F_{n,i}(t)$ is the CDF of values in sequence S_i.

Table IV. Summary of Distance Functions

Model	Distance Metrics
Click Sequence Model	*unigram*, *unigram+count*, *10gram*, *10gram+count*
Time-Based Model	*K-S* test
Hybrid Model	*5gram*, *5gram+count*

4.2.2. Applying Distance Functions to Clickstreams. Having defined three distance functions for computing sequence similarity, we now apply these metrics to our three clickstream models. Table IV summarizes the distance metrics we apply to each of our models. The Time-based model is the simplest case, because it only has one corresponding distance metric (K-S Test). For the Click Sequence and Hybrid models, we use several different parameterizations of our distance metrics.

Click Sequence Model. We use the common subsequence and common subsequence with counts metrics to compute distances in the CS model. However, these two metrics require that we choose N, the maximum number of k for k-gram subsequences to consider. We choose two values for N: 1 and 10, which we refer to as *unigram* and *10gram*. *Unigram* represents the trivial case of comparing common click events in two clickstreams while ignoring the ordering of clicks. *10gram* considers all k-grams where $k \leq 10$ (unigram, bigrams, trigrams, etc.). As shown in Table IV, we also instantiate *unigram+count* and *10gram+count*, which include the frequency counts of each unique subsequence.

Although values of $N > 10$ are possible, we limit our experiments to $N = 10$ for two reasons. First, given a clickstream of length n, the computational complexity of Ngram is $O(nN)$. When $N = n$, this overhead is significant considering that $O(n^2)$ subsequences will be computed for every user in a clickstream dataset. Second, long subsequences have diminishing utility, because they are likely to be unique for a particular user instead of representing common patterns. In our tests, we show that $N = 5$ already provides a good limit on computational overhead and subsequence overspecificity.

Hybrid Model. Like the Click Sequence model, distances between sequences in the Hybrid model can also be computed using the common subsequence and common subsequence plus count metrics. The only change between the Click Sequence and Hybrid models is that we must discretize the interarrival times between clicks so they can be encoded into the sequence. We do this by placing interarrival times into log-scale buckets (in seconds): [0, 1], [1, 10], [10, 100], [100, 1000], [1000, ∞]. Based on Figure 5, the interarrival time distribution is highly skewed, so log-scale buckets are better suited than linear buckets to evenly encode the times.

After we discretize the interarrival times and insert them into the clickstream, we use $N = 5$ as the parameter for configuring the two distance metrics. Further increasing N offers little improvement in the model but introduces extra computation overhead. As shown in Table IV, we refer to these metrics as *5gram* and *5gram+count*. Thus, each 5gram contains three consecutive click events along with two tokens representing interarrival time gaps between them.

4.3. Sequence Clustering

At this point, we have defined models of clickstreams as well as metrics for computing the distance between them. Our next step is to cluster users with similar clickstreams together. As shown in Section 3, Sybil and normal users exhibit very different behaviors, and should naturally form distinctive clusters.

To achieve our goal, we build and then partition a *clickstream similarity graph*. As shown in Figure 10, each user's clickstream is represented by a single node. The similarity graph is complete; that is, every pair of nodes is connected by a weighted edge, where the weight is the similarity distance between the sequences. Partitioning this graph means producing the desired clusters while minimizing the total weight of cut edges: users with similar activities (high weights between them) will be placed in the same cluster, while users with dissimilar activities will be placed in different clusters. The assumption is that the clustering process can separate Sybil from normal users. Note that not all Sybils and normal users exhibit homogeneous behavior; thus, we expect there to be multiple, distinct clusters of Sybils and normal users.

Graph Clustering. To cluster similarity graphs, we use METIS [31], a widely used multilevel k-way partitioning algorithm. The objective of METIS is to minimize the weight of edges that cross partitions. In the sequence similarity graph, longer distances (i.e., dissimilar sequences) have lower weights. Thus, METIS is likely to place dissimilar sequences in different partitions. METIS requires a parameter K that specifies the number of partitions desired. We will assess the impact of K on our system performance in Section 4.4. We choose METIS for its scalability. There are other fast clustering algorithms available such as DBSCAN [16], but this is nondeterministic and requires predefining a distance threshold. Note that METIS is a nonoverlapping clustering algorithm where one node can only be assigned to one cluster. We choose nonoverlapping clustering for its advantage of simplicity and the need for a binary decision in Sybil detection. An alternative approach is to use overlapping clustering algorithms where one node is assigned to multiple clusters. The label of the node then can be determined by a majority voting of all assigned clusters. Considering the complexity of overlapping clustering, we leave related experiments to future work.

Cluster Quality. A key question when evaluating our methodology is assessing the quality of clusters produced by METIS. In Section 4.4, we leverage our ground-truth data to evaluate false positives and negatives after clustering the sequence similarity graph. We label each cluster as "Sybil" or "normal" based on whether the majority of nodes in the cluster are Sybils or normal users. Normal users that get placed into Sybil clusters are false positives, while Sybils placed in normal clusters are false negatives. We use these criteria to evaluate different clickstream models and distance functions.

4.4. Model Evaluation

We now evaluate our clickstream models and distance functions to determine which can best distinguish Sybil activity patterns from those of normal users. We examine four different variables: (1) the choice of clickstream model, (2) the choice of distance function for each model, (3) what representation of clicks to use (specific activities or categories), and (4) K, the number of desired partitions for METIS.

Experiment Setup. The experimental dataset consists of 4,000 normal users and 4,000 Sybils randomly selected from our dataset. Given the large number of possible combinations of parameters, we used this sampled dataset to speed up the experiments. In each scenario, we build click sequences for each user (based on a given clickstream model and click representation), compute all distances between each pair of sequences, and then cluster the resulting sequence similarity graph for a given value of K. Finally, each experimental run is evaluated based on the false-positive and -negative error rates.

Model Analysis. First, we examine the error rates of different clickstream models and click representations in Figure 11. For the CS and Hybrid models, we encode clicks

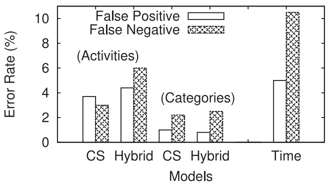


Fig. 11. Error rate of three models.

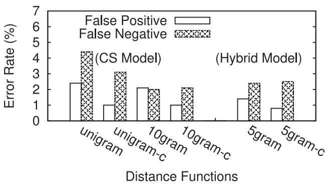


Fig. 12. Error rate using different distance functions.

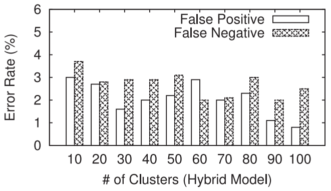


Fig. 13. Impact of number of clusters (K).

based on activities as well as categories. In the Time model, all clicks are encoded as interarrival times. In this experiment, we use *10gram+count*, *5gram+count*, and *K-S* as the distance function for CS, Hybrid, and Time, respectively. We fix $K = 100$. We investigate the impact of distance functions and K in subsequent experiments.

Two conclusions can be drawn from Figure 11. First, the CS and Hybrid models significantly outperform the Time-based model, especially in false negatives. Manual inspection of false-negative Sybils (about 10%) reveals that these Sybils click at a similar rate as normal users. Either these Sybils are operated by real people or the software that controls them has been intentionally rate limited. Despite the statistical difference between Sybils and normal users in click interarrival time (Figure 5), this metric alone is insufficient to disambiguate Sybils from normal users.

The second conclusion from Figure 11 is that encoding clicks based on category outperforms encoding by activity. This result confirms findings from the existing literature on web usage mining [1]: representing clicks using high-level categories (or *concepts*) instead of raw click types better exposes the browsing patterns of users. A possible explanation is that high-level categories have better tolerance for noise in the clickstream log. In the rest of our article, we use categories to encode clicks.

Next, we examine the error rate of different distance functions for the CS and Hybrid models. As shown in Figure 12, we evaluate the CS model using the *unigram* and *10gram* functions, as well as counting versions of those functions. We evaluate the Hybrid model using the *5gram* and *5gram+count* functions.

Several conclusions can be drawn from Figure 12. First, the *unigram* functions have the highest false-negative rates. This indicates that looking at clicks in isolation (i.e., without click transitions) is insufficient to discover many Sybils. Second, the counting versions of all three distance functions produce low false-positive rates. This demonstrates that the repeat frequency of sequences is important for identifying normal users. Finally, we observe that CS *10gram+count* and Hybrid have similar accuracy. This shows that click interarrival times are not necessary to achieve low error rates.

Finally, we examine the impact of the number of clusters K on detection accuracy. Figure 13 shows the error rate of Hybrid *5gram+count* as we vary K. The overall trend is that larger K produces lower error rates. This is because larger K grants METIS more opportunities to partition weakly connected sequences. This observation is somewhat trivial: if K equals the total number of sequences in the graph, then the error rate would be zero given our evaluation methodology. In Section 6, we discuss practical reasons K must be kept $\approx$100.

Summary. Our evaluation shows that the Click Sequence and Hybrid models perform best at disambiguating Sybils and normal users; *10gram+count* and *5gram+count* are the best distance functions for each model, respectively. We find that accuracy is highest when clicks are encoded based on categories, and when the number of partitions K is large. In the following sections, we will use these parameters when building our Sybil detection system.

5. INCREMENTAL SYBIL DETECTION

Our results in Section 4 showed that our models can effectively distinguish between Sybil clickstreams and normal user clickstreams. In this section, we leverage this methodology to build a real-time, incremental Sybil detector. This system works in two phases: first, we create clusters of Sybil and normal users based on ground-truth data, as we did in Section 4. Second, we compute the position of unclassified clickstreams in our sequence similarity graph. If an unclassified clickstream falls into a cluster representing clickstreams from ground-truth Sybils, we conclude the new clickstream is a Sybil. Otherwise, it is benign.

5.1. Incremental Detection

To classify a new clickstream given an existing clustered sequence similarity graph, we must determine how to "recluster" new sequences into the existing graph. We investigate three algorithms.

The first is *K Nearest Neighbor* (KNN). For a given unclassified sequence, we find the top-K nearest sequences in the ground-truth data. If the majority of these sequences are located in Sybil clusters, then the new sequence is classified as a Sybil sequence. For our experiment, we set to use the top three nearest sequences.

The second algorithm is *Nearest Cluster* (NC). We compute the average distance from an unclassified sequence to all sequences in each cluster. The unclassified sequence is then added to the cluster with the closest average distance. The new sequence is classified as Sybil or normal based on the cluster it is placed in.

The third algorithm is a less computationally intensive version of Nearest Cluster that we refer to as *Nearest Cluster-Center* (NCC). NC and KNN require computing the distance from an unclassified sequence to all sequences in the ground-truth clusters. We can streamline NC's classification process by precomputing *centers* for each cluster. In NCC, we only need to compute the distance from an unclassified sequence to the center of each existing cluster.

For each existing cluster, the center is chosen by *closeness centrality*. Intuitively, the center sequence is the one that has the shortest distance to all the other sequences in the same cluster. To be more robust, we precompute three centers for each cluster, that is, the three sequences with the highest closeness centrality.

5.2. System Evaluation

In this section, we evaluate our incremental Sybil detection system using our ground-truth clickstream dataset. We start by evaluating the basic accuracy of the system at classifying unknown sequences. Next, we evaluate how quickly the system can identify Sybils, in terms of the number of clicks in their clickstream. Finally, we explore how long the system can remain effective before it needs to be retrained using updated ground-truth data.

Detection Accuracy. We start with a basic evaluation of system accuracy using our ground-truth dataset. We split the dataset into training data and testing data. The training data contains 3,000 randomly selected normal users and 3,000 Sybils. The testing data contains the rest of the 2,998 normal users and 2,998 randomly selected Sybils. There is no overlap between training and testing data. We build sequence similarity graphs from the training data using the Hybrid model with *5gram+count* as the distance function. The number of clusters $K = 100$. In each sequence similarity graph, we label the Sybil and normal clusters.

Next, we examine the error rates of the incremental detector when unclassified users (2,998 Sybils and 2,998 normal users) are added to the sequence similarity graph. We perform this experiment three times, once for each of the proposed reclustering

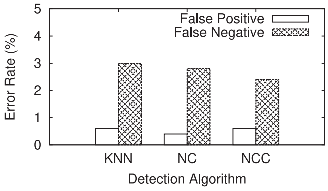


Fig. 14. Error rate of three reclustering algorithms.

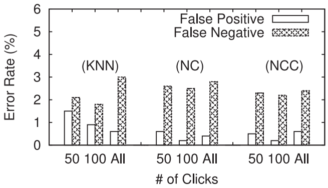


Fig. 15. Error rate versus maximum # of clicks in each sequence.

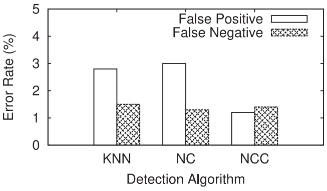


Fig. 16. Detection accuracy when training data is 2 weeks old.

algorithms (KNN, NC, and NCC). As shown in Figure 14, the error rates for all three reclustering algorithms are very similar, and all three have <1% false positives. NC has slightly fewer false positives, while NCC has the fewest false negatives.

Detection Speed. The next question we want to address is: *what is the minimum number of clicks necessary to accurately classify clickstreams?* Another way to frame this question is in terms of detection speed: *how quickly (in terms of clicks) can our system accurately classify clickstreams?* To identify and respond to Sybils quickly, we must detect Sybils using the minimal number of click events.

Figure 15 shows the results of our evaluation when the maximum number of clicks per sequence is capped in the testing data. The “All” results refer to a cap of infinity; that is, all clicks in our dataset are considered. Note that not all sequences in our dataset have 50 or 100 clicks: some Sybils were banned before they produced this may clicks. Hence, the caps are upper bounds on sequence length.

Surprisingly, the “All” results are not the most accurate overall. As shown in Figure 15, using all clicks results in more false negatives. This occurs due to overfitting: given a large number of very long clickstreams from normal users, it is likely that they will occasionally exhibit unusual, Sybil-like behavior. However, this problem is mitigated if the sequence length is capped, since this naturally excludes these infrequent, aberrant clickstreams.

In contrast to the “All” results, the results from the ≤50 click experiments produce the most false positives. This demonstrates that there is a minimum sequence length necessary to perform accurate classification of clickstreams. We repeated these experiments using CS/*10gram+count* and received a similar result, which we omit for brevity.

There are two additional, practical take-aways from Figure 15. First, the NCC algorithm performs equally well versus NC and KNN. This is a positive result, since the computational complexity of NCC is dramatically lower than NC and KNN. Second, we observe that our system can produce accurate results (false positives <1%, false negatives <3%) when only considering short sequences. This means that the system can make classifications quickly, without needing to store very long clickstreams in memory.

Accuracy Over Time. In order for our incremental detection system to be practically useful, its accuracy should remain high for long periods of time. Put another way, sequence similarity graphs trained with old data should be able to detect fresh Sybil clickstreams. To evaluate the accuracy of our system over time, we split our dataset based on date. We train our detector using the early data, and then apply the detector to the later data. We restrict our analysis to data from April 2011; although we have Sybil data from March 2011, we do not have corresponding data on normal users for this month.

Figure 16 shows the accuracy of the detector when it is trained on data from March 31 to April 15, then applied to data from April 16 to 30. As the results show, the detector remains highly accurate for at least 2 weeks after it has been trained using the NCC reclustering algorithm. Unfortunately, the limited duration of our dataset prevents us from examining accuracy at longer time intervals.

We repeated this experiment using only 1 week of training data, but the false-negative rate of the detector increased to ≈10%. This shows that the detector needs to be trained on sufficient data to provide accurate results. Note that the age of the accounts may also affect the detection accuracy over time. Unfortunately, we don't have the information on account age to analyze the edge effects.

6. SEMISUPERVISED SYBIL DETECTION

Our incremental Sybil detection system from Section 5 has a serious shortcoming: it must be trained using large samples of ground-truth data. In this section, we develop a semisupervised Sybil detection system that requires only a small, constant amount of ground truth. The key idea is to build a clustered sequence similarity graph as before. But instead of using full ground truth of all clickstreams to mark a cluster as Sybil or normal, we only need a small number of clickstreams of known real users as "seeds" that color the clusters they reside in. These seeds can be manually verified as needed. We *color* all clusters that include a *seed* sequence as "normal," while uncolored clusters are assumed to be "Sybil." Since normal users are likely to fall under a small number of behavioral profiles (clusters in the graph), we expect a small fixed number of seeds will be sufficient to color all clusters of normal user clickstreams. We explicitly don't use known Sybils as seeds because they will optimize the model to only capture existing (or outdated) types of attacks. Instead, we use real users as seeds to model normal behavior, which has the chance to capture new attacks based on "abnormal" activities.

Figure 17 depicts our semisupervised approach, showing how METIS partitions nodes into clusters that are then colored if they contain seed users. Once the system is trained in this manner, it can be used incrementally to detect more Sybils over time, as described in Section 5.

In this section, we discuss the design of our unsupervised system and evaluate its performance. We begin by analyzing the number and composition of seeds that are necessary to ensure high accuracy of the system. Next, we evaluate the performance of the system by comparing its accuracy to our ground-truth data. Finally, we examine how the ratio of Sybils to normal users in the unclassified data impacts system accuracy.

6.1. Seed Selection and Composition

Number of Seeds. The most important parameter in our unsupervised Sybil detection system is the number of seeds. On one hand, the number of seeds needs to be large and diverse enough to color all "normal" clusters. Normal clusters that remain uncolored are potential false positives. On the other hand, the seed set needs to be small enough to be practical. If the size of the seed set is large, it is equivalent to having ground truth about the dataset, which is the situation we are trying to avoid.

We now conduct experiments to determine how many seeds are necessary to color the clusters. We choose 3,000 Sybils and 3,000 normal users at random from our dataset to be the unclassified dataset. We also randomly choose some number of additional normal users to be the seeds. As in Section 5, we use the Hybrid model with the *5gram+count* distance function. We also conducted experiments with CS/*10gram+count*, but the results are very similar and we omit them for brevity.

Figure 18 depicts the percentage of normal clusters that are correctly colored for different values of K (number of METIS partitions) as the number of seeds is varied. As expected, fewer seeds are necessary when K is small because there are fewer clusters

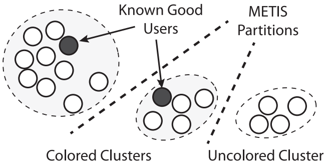


Fig. 17. Unsupervised clustering with coloring.

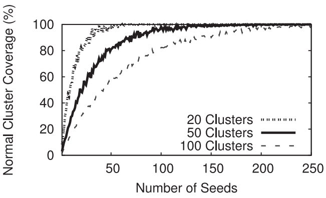


Fig. 18. Number of seeds versus percent of correctly colored normal user clusters.

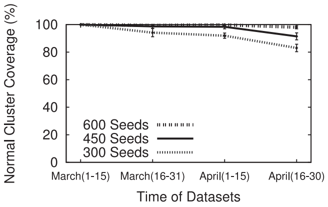


Fig. 19. Consistency over time of normal seeds for coloring.

(and thus each cluster includes more sequences). When $K = 100$, 250 seeds (or 4% of all normal users in the experiment) are able to color 99% of normal clusters.

Seed Consistency over Time. Next, we examine whether a set of seeds chosen at an early date is equally effective at coloring clusters based on later data. In other words, we want to know if the seeds are consistent over time. If this is not the case, it would represent additional overhead on the deployment of our system.

To test seed consistency over time, we divide our 2 months of Sybil clickstream data into four 2-week-long datasets. We add an equal number of randomly selected normal users to each of the four datasets. Finally, we select an additional x random normal users to act as seeds. We verify (for each value of x) that these seeds color 100% of the normal clusters in the first temporal dataset. We then evaluate what percentage of normal clusters are colored in the subsequent three temporal datasets. In all experiments, we set $K = 100$, that is, the worst-case scenario for our graph coloring approach.

The results of the temporal consistency experiments are shown in Figure 19. In general, even though the Sybil and normal clickstreams change over time, the vast majority of normal clusters are successfully colored. Given 600 seeds, 99% of normal clusters are colored after 4 weeks, although the percentage drops to 83% with 300 seeds. These results demonstrate that the seed set does not need to be drastically altered over time.

6.2. Coloring Evaluation

We now evaluate the overall effectiveness of our Sybil detection system when it leverages unsupervised training. In these experiments, we use our entire clickstream dataset. We choose x random normal users as seeds, build and cluster the sequence similarity graph using Hybrid/*5gram+count*, and then color the clusters that contain the seeds. Finally, we calculate the false-positive and -negative rates using the same methodology as in Section 5, that is, by comparing the composition of the colored clusters to ground truth.

The results are shown in Figure 20. As the number of seeds increases, the false-positive rate decreases. This is because more seeds mean more normal clusters are correctly colored. With just 500 seeds, the false-positive rate drops to 0.7% (false-positive rate is 0.5% for 600 seeds). Unfortunately, relying on unsupervised training does increase the false-negative rate of our system by 2% versus training with ground-truth data. However, in cases where ground-truth data is unavailable, we believe that this is a reasonable tradeoff. Note that we also repeated these experiments with CS/*10gram+count*, and it produced slightly higher false-positive rates, although they were still <1%.

Unbalanced Training Dataset. Next, we evaluate the impact of having an unbalanced training dataset (e.g., more normal users than Sybils) on the accuracy of our

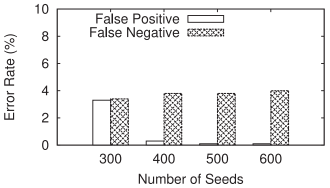


Fig. 20. Detection accuracy versus number of seeds.

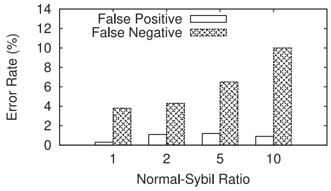


Fig. 21. Detection accuracy versus normal-Sybil ratio.

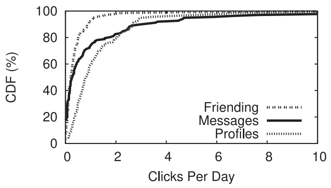


Fig. 22. Clicks per day by outlier normal users.

system. Thus far, all of our experiments have assumed a roughly equal percentage of Sybils and normal users in the data. However, in practice, it is likely that normal users will outnumber Sybils when unsupervised learning is used. For example, Facebook suspects that 8.7% of its user base is illegitimate, out of >1 billion total users [12].

We now evaluate how detection accuracy changes when we decrease the percentage of Sybils in the training data. In these experiments, we construct training sets of 6,000 total users with different normal-to-Sybil ratios. We then run unsupervised training with 500 seeds (based on Figure 20). Finally, we incrementally add an additional 2,998 Sybils and 2,998 normal users to the colored similarity graph using the NCC algorithm (see Section 5.1). We ran additional tests using the NC and KNN algorithms, but the results were very similar and we omit them for brevity.

Figure 21 shows the final error rate of the system (i.e., after testing data has been incrementally added) for varying normal-to-Sybil ratios. The false-positive rate remains $\leq$1.2% regardless of the normal-to-Sybil ratio. This is a very good result: even with highly skewed training data, the system is unlikely to penalize normal users. Unfortunately, the false-negative rate does rise as the number of Sybils in the training data falls. This result is to be expected: the system cannot adequately classify Sybil clickstreams if it is trained on insufficient data.

Handling False Positives. The previous analysis demonstrates that our system achieves high accuracy with a false-positive rate of 1% or less. Through manual inspection, we find that "false positives" generated by our detector exhibit behaviors generally attributed to Sybils, including aggressively sending friend requests or browsing profiles. In real-world OSNs, suspicious users identified by our system could be further verified via existing complementary systems that examine other aspects of users. For example, this might include systems that classify user profiles [53, 66], systems that verify user real-world identity [17], or even Sybil detection systems using crowdsourced human inspection [59]. These efforts could further protect benign users from misclassification.

Model Updating over Time. In practice, we foresee that our model needs to be updated periodically to capture dynamic changes of attackers and handle compromised accounts. Our ground-truth dataset does not differentiate compromised accounts with Sybils as long as they are controlled by the attacker. In practice, compromised accounts are likely to have legitimate historical clickstreams, which can be confusing to the detector. By building new models periodically, we can mitigate this issue with the more recent clickstream data.

7. PRACTICAL SYBIL DETECTION

In this section, we examine the practical performance of our proposed Sybil detection system. First, we shipped our code to the security teams at Renren and LinkedIn, where it was evaluated on fresh data in a production environment. Both test results are very positive, and we report them here. Second, we discuss the fundamental limits of our approach, by looking at our impact on Sybil accounts that can perfectly mimic the clickstream patterns of normal users.

7.1. Real-World Sybil Detection

With the help of supportive collaborators at both Renren and LinkedIn, we were able to ship prototype code to the security teams at both companies for internal testing on fresh data. We configured our system to use semisupervised learning to color clusters. Sequence similarity graphs are constructed using the Hybrid model and the *5gram+count* distance function, and the number of METIS partitions K is 100.

Renren. Renren's security team trained our system using clickstreams from 10K users, of which 8K were randomly selected, and 2K were previously identified as suspicious by the security team. These clickstreams were collected between January 17 and 27, 2013. Five hundred honest users that have been manually verified by Renren's security team were used as seeds. Once trained, our system was fed clickstreams from 1,000,000 random users (collected in early February 2013) for classification as normal or suspicious. In total, our system identified 22K potential Sybil accounts.

While corporate privacy policies prevented Renren from sharing detailed results with us, their feedback was very positive. They indicated that our system identified a *new type of attack* performed by a large cluster of "image spammers." These accounts' clickstream behavior focused heavily on photo sharing. Manual inspection revealed that these accounts embedded spammy text and URLs to promote brands of clothes and shoes. Traditional text analysis-based spam detectors and URL blacklists were unable to catch this new attack since the content was embedded into images. Our system identified it immediately for their abnormal clickstream behavior compared to normal users. A dedicated engineering team has taken over our code for production-level implementation. Renren's corporate policy prevents the team from revealing further implementation details and detection results.

LinkedIn. LinkedIn's security team used our software to analyze the clickstreams of 40K users, of which 36K were randomly sampled, and 4K were previously identified as suspicious by the security team. These clickstreams were gathered in February 2013. Again, our feedback was very positive, but did not include precise statistics. However, we were told that our system confirmed that ≈1,700 of the 4,000 suspicious accounts are likely to be Sybils. Our system also detected an additional 200 previously unknown Sybils.

A closer look at the data shows that many of the accounts not detected by our system were borderline accounts with specific flags popping up in their profiles. For example, some accounts had unusual names or occupational specialties, while others had suspicious URLs in their profiles. These results remind us that a user behavior model is clearly only a part of the equation. For example, clickstream analysis does not examine the content that users posted, and may miss Sybils that only exhibit anomalies in content. To this end, our system should be used in conjunction with existing profile analysis tools and spam detectors [3, 18, 58, 59, 67].

7.2. Limits of Sybil Detection

Finally, we wish to discuss the worst-case scenario for our system, that is, a scenario where attackers have full knowledge of the clickstream patterns for real users and are able to instrument the behavior of their Sybils to mimic them precisely. In this attack model, the attacker's goal is to have Sybils carry out malicious actions (e.g., sending spam) without being detected. However, to evade detection, these Sybils must limit themselves to behavior consistent with that of normal users.

We can thus bound the capabilities of Sybils that avoid detection in this attack model. First, the Sybil's clickstream must remain inside the "normal" clusters produced by our detector. Second, the most aberrant behavior within a given "normal" cluster is exhibited by real users within the cluster who are farthest from the center. The activities performed by these *outliers* serve as effective bounds on Sybil behavior. Sybil clickstreams cannot deviate from the center of the cluster more than these outliers; otherwise, they will be excluded from the cluster and risk detection. Thus, we can estimate the maximum amount of malicious activity a Sybil could perform (without getting caught) by studying these outliers.

We now examine the behavior of outliers. We calibrate our system to produce clusters with false-positive rate $<1\%$ using Hybrid/*5gram+count*, and $K = 100$. In this configuration, the detector outputs 40 Sybil and 60 normal clusters when run on our full dataset. Next, we identify the two farthest outliers in each normal cluster. Finally, we plot the clicks per day in three activities from the 120 outliers in Figure 22. We focus on clicks for sending friend requests, posting status updates/wall messages, and viewing user profiles. These activities correspond to the three most common attacks observed in our data, that is, sending friend request spam, status/wall spam, and profile crawling.

As shown in Figure 22, 99% of outliers generate $\leq$10 clicks per day in the target activities. In the vast majority of cases, even the outliers generate less than two clicks per day (reflecting bursty usage). These results show that the effective bound on Sybil behavior is very tight; that is, to avoid detection, Sybils can barely generate any clicks each day. These bounds significantly increase the cost for attackers who will have to use many more Sybils to maintain the same level of spam generation capacity.

8. UNSUPERVISED USER BEHAVIOR MODELING

Thus far, we have demonstrated the effectiveness of clickstream analysis for Sybil detection. However, the current model oversimplifies the differences among users with a binary classification (i.e., either malicious or benign). It helps to capture attackers but does not provide explicit knowledge and interpretation about how users (or attackers) behave and how their behavior changes over time. For instance, among the Sybil accounts, there are likely different attacking strategies used by different attackers, while the current model cannot differentiate or explain them.

To this end, we extend this clickstream model beyond binary classification to identifying and understanding fine-grained user behaviors in online services. Our goal is to identify prevalent user behaviors in a given service without any prior knowledge or labels (*unsupervised*). In addition, we seek to provide interpretable results/interfaces to help service providers understand captured user behavior.

At the high level, our system assumes that human behavior naturally forms clusters. Despite users' differences in personalities and habits, their behavioral patterns within a given service cannot be entirely disparate. Our goal is to identify such natural clusters as behavioral models. In addition, user behavior is likely multidimensional. We expect user clusters to fall into a tree hierarchy instead of a one-dimensional structure (Figure 23). In this hierarchy, most prominent features are used to place users into high-level categories, while less significant features characterize detailed substructures.

We build our system with a new algorithm to capture hierarchical clickstream clusters, called *iterative feature pruning*. We use the new algorithm to replace METIS to partition the similarity graph and identify clusters of users with similar clickstream activities. To capture the hierarchical structure, we recursively partition newly generated clusters while *pruning* the feature set used to measure clickstream similarity.

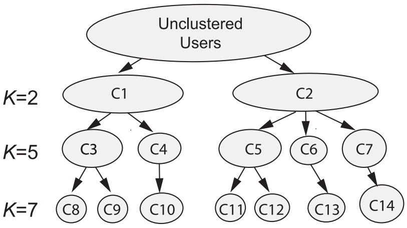


Fig. 23. Hierarchy of the behavioral clusters.

Intuitively, by identifying and pruning dominating features in higher-level clusters, we allow the secondary features to manifest and discover more fine-grained subclusters. Also, the pruned features are indicative of why this cluster is formed, which can help service providers to understand the behavioral model.

In this section, we first describe the feature-pruning algorithm to identify clusters in the clickstream similarity graph. Then, we build a visualization tool to help service providers examine and understand behavioral clusters. We will evaluate the system with a user study (Section 9) and selected case study analysis (Section 10). We then discuss the quality of results (Section 11) and scalability issues (Section 12).

8.1. Feature-Pruning-Based Clickstream Clustering

A similarity graph dominated by very few features gives little insight on subtle differences between users. The generated clusters may only describe the broadest user categories, while interesting and detailed behavioral patterns remain hidden. We recognize that the similarity graph has the capability to capture user behavior at different levels of granularity. We implement *iterative feature pruning* as a means of identifying fine-grained behavioral clusters within existing clusters and recursively partitioning the similarity graph. In the following, we first introduce the key steps of our clustering algorithm and feature pruning. Then we describe using pruned features to interpret the meaning of the clusters and the technical details to determine the number of clusters.

Iterative Feature Pruning & Clustering. We explain how our algorithm works using the example in Figure 23. We start with a similarity graph of all users, where clickstream similarity is measured based on the full feature set (union of all *k*-grams). By partitioning the similarity graph, we get the top-level clusters C_1 and C_2. The partitioning algorithm we use is Divisive Hierarchical Clustering [32], which can work on arbitrary metric space and find clusters of arbitrary shapes.

To identify more fine-grained subclusters within C_1 or C_2, we perform feature pruning: we identify the primary features that are responsible for forming the parent cluster, remove them from the feature set, and use the remaining secondary features to further partition the parent. For example, suppose C_1 is the current parent cluster. We first perform feature selection to determine the key features (i.e., k-grams) that classify users into C_1. Then, to partition C_1, we remove those top *k*-grams from the feature set and use the remaining *k*-grams to compute a new similarity graph for C_1. In this way, secondary features can step out to partition C_1 into C_3 and C_4 (by running Divisive Hierarchical Clustering on the new similarity graph). For each of the newly generated clusters (e.g., C_3 and C_4), we recursively run the same process to produce more fine-grained subclusters. Our algorithm stops when all the new partitions cannot be further split; that is, clustering quality reaches a minimal threshold. The result is a tree hierarchy of behavioral clusters.

The key step of feature pruning is finding the primary features responsible for forming the parent cluster. We select features based on Chi-square statistics (χ^2) [65], a classic metric to measure features' discriminative power in separating data instances of different classes. For a given cluster (e.g., C_1), we compute a χ^2 score for each feature. The χ^2 score measures the difference between the distribution of this feature value in C_1 and the distribution of this feature outside of C_1. We sort and select the top features with the highest χ^2 scores, which are the key features to distinguish users in C_1 from the rest of users. Our empirical data shows that χ^2 distribution usually exhibits a "long-tail" property—only a small number of dominating features have high χ^2 scores. We automatically select the top features by identifying the sweet point (or turning point) in the χ^2 distribution [46]. An example is shown in Figure 26.

Understanding the Behavioral Clusters. We can infer the meaning of the clusters based on the selected features during the feature pruning phase. A feature is selected because users in this cluster are distinct from users outside the cluster on this particular feature dimension. Thus, it can serve as an explanation for why a cluster has formed and which user behaviors the cluster encompasses. Later we construct a visualization tool to help service providers interpret behavioral clusters.

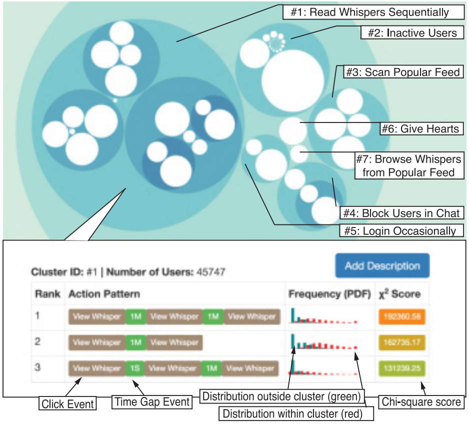
Determining the Number of Subclusters. For each parent cluster (and its similarity graph), our system identifies the natural number of subclusters within. To do so, we monitor the changes of the overall *clustering quality* while continuously partitioning the graph to more subclusters. We stop when generating more subclusters will no longer improve the clustering quality. The metric to assess clustering quality is the widely used *modularity*, which measures the density of edges inside clusters to edges outside clusters [8]. Modularity $Q = \frac{1}{2m}\sum_{ij}[W_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j)$, where W_{ij} is the edge weight (similarity) between user i and j, k_i and k_j are their weighted degrees, m is the total number of edges in the graph, and $\delta(c_i, c_j) = 1$ if i and j belong to the same cluster (otherwise, the value is 0). The modularity value ranges from -1 to 1, with a higher value indicating a better clustering quality.

8.2. Cluster Visualization

We then build a visualization tool for service providers to examine and understand user behavioral clusters generated by our algorithm. The tool allows service providers to answer key questions about their users. For example, what are the major behavioral categories? Which behavior is more prevalent? What's the relationship between different types of behavior?

Visualization Interface. Figure 24 shows a screenshot of our tool displaying the behavioral clusters of Whisper (best viewed in color). We built this tool using *D3.js*, a JavaScript library for data visualization. By default, we display the cluster hierarchy using Packed Circle [62], where child clusters are nested within their parent cluster. This gives a clear view of the hierarchical relationships of different clusters. Circle sizes reflect the number of users in the cluster, which allows service providers to quickly identify the most prevalent user behaviors. Finally, the visualization tool is zoomable and easy to navigate among clusters. We also implemented other interfaces such as Treemaps, Sunburst, and Icicle. Service providers can choose any of these based on personal preference. We use Packed Circle as default because it leaves more space between clusters, making it easier to visually separate different clusters.

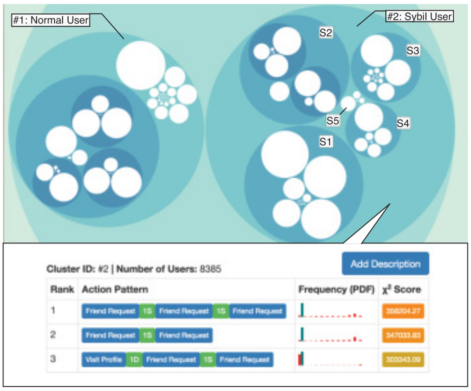
To understand the user behavior in a specific cluster, we can click the cluster to pop up an information window. Take the one in Figure 24 for example: we show the basic cluster information on top, including clusterID and the number of users. Below that is a list of "Action Patterns" (k-grams) selected by our Feature Pruning algorithm to describe how users behave. Each row contains one pattern, ranked by χ^2 score (brighter color indicates higher score). The "Frequency (PDF)" column shows how frequently each



Cluster ID: #1 | Number of Users: 45747

Rank	Action Pattern	Frequency (PDF)	χ^2 Score
1	View Whisper 1M View Whisper 1M View Whisper		192360.58
2	View Whisper 1M View Whisper		162736.17
3	View Whisper 1S View Whisper 1M View Whisper		131239.25

Fig. 24. Whisper behavioral clusters. Cluster labels are manually input based on results of each cluster. The pop-up window shows users in Cluster #1 tend to sequentially read whispers.



Cluster ID: #2 | Number of Users: 8385

Rank	Action Pattern	Frequency (PDF)	χ^2 Score
1	Friend Request 1S Friend Request 1S Friend Request		358204.27
2	Friend Request 1S Friend Request		347033.83
3	Visit Profile 1D Friend Request 1S Friend Request		303343.09

Fig. 25. Renren behavioral clusters. The pop-up window shows users in Cluster #2 focus on sending friend requests and browsing user profiles.

Table V. Clustering Results for Renren and Whisper Data

Dataset	# Clusters (Leaf)	Depth	Total Features	# Selected Features	Selected Feature Per Cluster
Renren	108 (95)	4	80,903	409	5.42
Whisper	107 (95)	4	66,098	592	6.81

action pattern appears among users of this cluster. The red bar indicates the pattern frequency (probability density function) inside the cluster, and the green bar denotes frequency outside of this cluster. Intuitively, the more divergent the two distributions are, the more distinguishing power the pattern has. In this example, the first pattern shows users viewing whispers sequentially with a time interval of 1 minute or less. The red bar is much more skewed to the right, indicating that users in this cluster perform this action more often than users outside. Finally, service providers can "add descriptions" to the cluster using the button in blue.

8.3. Whisper and Renren Clusters

Configurations. We run our system on Whisper and Renren datasets and display the behavioral clusters in Figures 24 and 25. We apply the same configuration on both runs: the partitioning of a cluster stops if the modularity reaches a threshold 0.01 (insignificant cluster structure). We intentionally set a loose threshold to let the algorithm dig out very detailed subclusters. In practice, service providers can tune this parameter depending on how detailed they need the behavioral clusters to be. Details regarding algorithm implementation and complexity are explained in Section 11.

Clustering Results. The high-level clustering results are shown in Table V. For the Renren dataset, the resulting hierarchy contains 108 clusters (95 leaf clusters). For Whisper, our system produces a tree hierarchy of 107 clusters (root included) with 95 leaf clusters. The depths of both trees are 4.

Recall that our system selects key features for each cluster to help the interpretation of corresponding user behaviors. A quick analysis shows that Chi-square statistics for features in the clusters are *highly skewed*, meaning that only a few features are strongly associated with the respective cluster. An example is shown in Figure 26. The

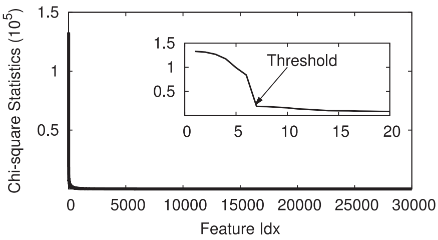


Fig. 26. The distribution of Chi-square statistics in an example Whisper cluster.

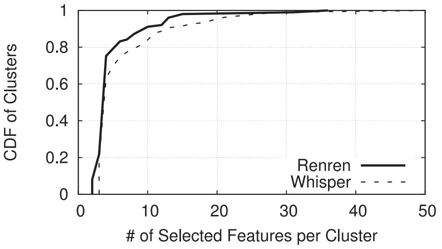


Fig. 27. Number of selected features per cluster for Renren and Whisper datasets.

distribution is clearly a long-tail distribution with a "turning point" at the 7_{th} feature. We can automatically select the top features by identifying the turning point in the curve using the algorithm of [46].

As shown in Table V, our feature selection reduces the feature space by orders of magnitude. For instance, the Whisper data originally contains 80,903 unique kgrams as features. After clustering, we only have on average 6.81 features per cluster to characterize the user behavior in the cluster. As shown in Figure 27, 80% of the clusters have fewer than five selected features, and 90% of the clusters have fewer than 10. This makes it possible for people to understand the cluster without looking through the full feature set. Our visualization tool also exclusively displays the selected features in the pop-up window, so that people can focus on key features on user behaviors without being distracted by the long tail of less important features.

9. EVALUATION: CLUSTER LABELS

In the following, we analyze the behavioral clusters in Whisper and Renren and demonstrate their effectiveness in identifying unexpected behavior and predicting future activities. Our evaluation contains three steps. First, to evaluate the ease of understanding and labeling behavioral clusters, we run a user study. We ask the participant to read cluster information and describe the corresponding user behavior. Then we examine whether different people give consistent descriptions. Second, we perform in-depth case studies on the unusual behavioral clusters and provide new insights to both networks. Third, we evaluate cluster quality, that is, how well behavioral clusters capture similar users.

9.1. User Study to Interpret Clusters

User behavioral models need to be intuitive and understandable to the service providers. Thus, we conduct a user study to answer two key questions: Are these behavioral clusters understandable to humans? How consistently do different people interpret the corresponding user behaviors?

In this user study, we ask participants to browse behavioral clusters using our visualization tool (Packed Circle interface). For each cluster, the participant is asked to describe the user behavior using his or her own words (in one sentence) based on the information displayed. If a cluster is not understandable to the participant, he or she can mark it as "N/A." Since our tool is designed for service providers, we expect they will have basic technical backgrounds. Our participants include 15 graduate students in computer science who have basic knowledge in online social networks. Our experiment is likely to represent a lower bound of performance, since real system admins are likely to have a more in-depth knowledge about their services. To best utilize participants' time, we only use the Whisper clusters (Figure 24), and the participants only look at

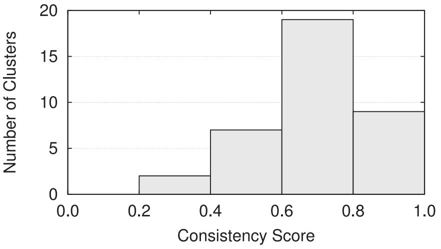


Fig. 28. Distribution of consistency score.

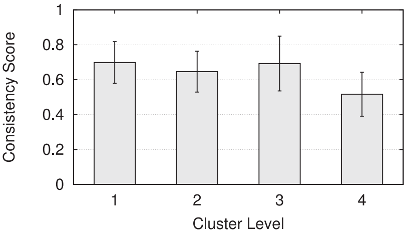


Fig. 29. Consistency score versus cluster level.

top clusters that cover 90% of users at each level of the hierarchy (37 clusters in total). Before the test, we ask the participants to use the Whisper app for at least 10 minutes to get familiar with it. Each participant also goes through a quick instruction session to learn how to use the visualization tool and how to read the information in the pop-up window.

9.2. User Study Results

We gathered a total of 555 descriptions from the participants on the 37 clusters (15 descriptions per cluster). We find that the behavioral clusters are generally understandable to the participants. Out of the 555 descriptions, 530 (95.5%) are valid descriptions about user behaviors (others are "N/A" marks). In addition, most participants can finish the task within a reasonable amount of time. The average completion time is 28.7 minutes (46 seconds per cluster).

To understand the consistency of the descriptions, we let three external experts independently read and assess the collected descriptions. These experts are graduate students recruited outside of our research group and none of them participated in labeling clusters in the first round. For each cluster, an expert reads all 15 descriptions and selects the largest group of consistent descriptions. Two descriptions are considered "consistent" if they are describing semantically the same aspect of user behavior. Then we compute a consistency score (0 to 1), which is the ratio of the maximum number of consistent descriptions over all descriptions. For example, if 10 out of the 15 descriptions are consistent, the score is $10/15 = 0.667$. The final consistency score is averaged over three experts. Figure 28 shows the consistency score distribution. The most common scores range from 0.6 to 0.8. The score distribution skews heavily to the right. This indicates that most clusters can be interpreted consistently.

Upon examining clusters with low consistency scores, we have two key observations. First, lower-level clusters are more difficult to interpret. As shown in Figure 29, average consistency scores decrease as we move further along the tree hierarchy. Intuitively, lower-level clusters represent more specific or even outlier behavior that is difficult to describe. Second, we find clusters with more "complex" features are harder to interpret. We perform correlation analysis between the consistency score and the number of unique event types in selected features, and find they correlate negatively (Pearson coefficient $r = -0.4$, $p = 0.02$). We also manually examined the few clusters that received "N/A" marks. These clusters often exhibit mixed behaviors (e.g., users are engaged in both posting whispers and sending hearts), which are difficult to interpret for some participants.

Finally, regarding the correctness of the descriptions, the authors manually examined the cluster descriptions and validated them based on the distributions of the top behavioral features in each cluster. We will present more detailed case studies in the next section. We also add short labels to the top-level clusters in Whisper and Renren

Table VI. Users Becoming Dormant Over Time: We Split the Clickstream Data into Three Snapshots and Report the Number of Users Who Migrate to the Dormant Cluster Over Two Adjacent Snapshots

	# (%) of Users Migrating to the Dormant Cluster	
Cluster	Snap 1 → Snap 2	Snap 2 → Snap 3
Dormant Cluster	15,873/16,872 (94%)	16,161/16,314 (99%)
Semidormant Clusters	363/9,383 (4%)	2,026/11,773 (17%)
Other Clusters	63/73,735 (0.09%)	804/71,903 (1.1%)

based on the descriptions from the user study and our own interpretations. The labels are shown in Figure 24 and Figure 25, respectively.

10. EVALUATION: CASE STUDIES

Next, we present in-depth analysis on a few behavioral clusters as case studies. We have two goals. First, by analyzing the user behavior in these clusters, we validate the correctness of our cluster labels. Second, we explore the interesting (or unexpected) user behavior and demonstrate the prediction power of the user behavioral models. Due to space limitations, we focus on two clusters from Whisper (Cluster #2 and Cluster #4) and one from Renren (Sybil Cluster).

10.1. Case Study 1: Inactive Whisper Users

We start with Cluster #2, which is labeled as inactive users. The selected action patterns of this cluster consist almost entirely of "receiving notification" events, indicating these users have not been actively engaged with the app. This is also confirmed in Figure 30: users in Cluster #2 have far fewer active days (when users actively generate clicks) than the rest of the users. A remarkable 80% of users in Cluster #2 did not generate any active events through the 45 days, representing completely dormant users. In fact, our algorithm successfully groups dormant users into a separate subcluster (Figure 24, the biggest subcluster in Cluster #2).

Contrary to expectations, inactive users are not outliers. Cluster #2 is the second-largest cluster with 21,962 users (20% of all users). From the perspective of service providers, it is important to identify the early signals of user disengagement and implement mechanisms to regain user activities.

Predicting Dormant Users. We demonstrate the effectiveness of our behavioral models in predicting future user dormancy. The high-level idea is simple: Whisper can build behavioral models using users' most recent clickstreams and update the models at regular intervals (e.g., every month). Our hypothesis is that users placed in the "inactive" cluster are more likely to turn completely dormant. Thus, we can use the inactive cluster to predict future dormant users.

We validate this hypothesis by investigating whether users in the "inactive" cluster will migrate to the "dormant" cluster over time. To do so, we split our clickstream data by date into three snapshots: Oct. 13 to 27, Oct. 28 to Nov. 12 and Nov. 13 to 26. Then we generate behavioral clusters for each snapshot. The inactive cluster can be easily pinpointed within each snapshot based on selected activity patterns (i.e., notification events). Also, we consistently find the following substructures within the inactive cluster: a big "dormant" cluster in which users have zero active events, alongside several "semidormant" clusters in which users are occasionally active.

In Table VI, we compare clusters from two adjacent snapshots to determine the likelihood of users migrating into the dormant cluster. The results confirm our hypothesis: users in semidormant clusters are more likely to migrate to the dormant cluster than others. For example, 17% of semidormant users in snapshot 2 end up in the dormant

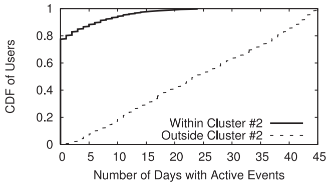


Fig. 30. Number of days when users have active events in clickstreams.

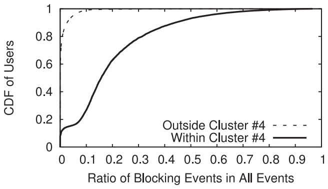


Fig. 31. Ratio of blocking event over all events in a clickstream.

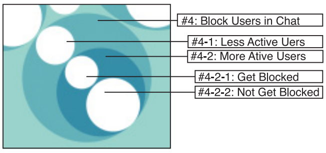


Fig. 32. The subclusters within Cluster #4.

Table VII. Activity Statistics for Users Inside and Outside Cluster #4 (*The Difference Is Statistically Significant Based on Welch Two-Sample t-Tests)

	Statistics: Mean (STD)		T-Statistics (p Value)
Actions per Day	Inside C#4	Outside C#4	In Versus Out
Whisper Posted	1.25 (1.77)	0.65 (1.46)	27.43 (p<0.001)*
Replies Received	0.70 (4.09)	0.26 (1.41)	8.89 (p<0.001)*
Heart Received	2.39 (48.68)	0.69 (5.34)	2.93 (p=0.0034)*
Chats Initiated	2.20 (10.93)	1.18 (3.98)	7.79 (p<0.001)*

Table VIII. Top Whisper Keywords for Users in Cluster #4 and Users Outside Cluster #4

Users	Top 30 Keywords
Inside C#4	20f, 19f, 18f, 17f, 29, f, roleplay, daddy, wet, role, lesbians, 17,lesbian, kinky, trade, bored, kik, weakness, nude, threesome,bestfriend, msg, shower, boys, chubby, nipples, horny, female, dirty, message
Outside C#4	religion, que, bullshit, 18m, personally, bible, eventually, faith,sign, plenty, hilarious, congratulations, gender, brain, idiot,dumbass, ignorant, quite, depends, animals, google, society, loss,count, health, sexuality, em, business, sound, foot

cluster in snapshot 3, while only 1.1% of other users do so. Users already within the dormant cluster are highly likely to remain there through future snapshots (94%–99%). This result shows that our behavioral models can successfully track and predict the dormancy of Whisper users. It allows service providers to make timely interventions before losing user participation.

10.2. Case Study 2: Hostile Behaviors of Whisper Chatters

Next, we analyze Cluster #4, which contains 7,026 users who tend to block other people during private chat. As shown in Figure 31, users in this cluster perform blocking actions much more frequently. Eighty percent of users spend more than 10% of their total clicks on blocking events. In contrast, only 1% of users outside Cluster #4 achieve this ratio.

Next, we explore the possible causes of the blocking events. A private chat is initiated by the user who wants to talk to whisper authors. Our hypothesis is that users in Cluster #4 are more likely to post whispers that attract unwanted chatters to harass them. To validate this, we list behavioral statistics for users inside and outside Cluster #4 in Table VII. Users in Cluster #4 are more active in posting public whispers, which attract more hearts and replies from others (statistically significant based on Welch t-tests). These users are likely to experience harassment as a side effect.

Users may attract unwanted chat messages due to the topics they write about. We analyze users' whisper content in Cluster #4 and find they often consist of sexually explicit messages (sexting). Table VIII lists the top keywords from users in and outside

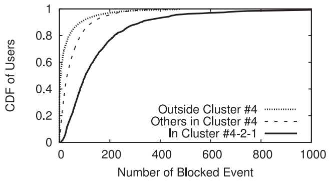
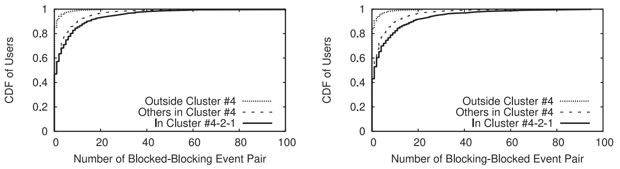


Fig. 33. Number of being-blocked events per user.



(a) Blocked→Blocking Event (b) Blocking→Blocked Event

Fig. 34. Number of paired blocking and blocked events per user.

Cluster #4. Keywords are ranked based on how strongly they are associated with the cluster. For each keyword, we compute a simple correlation ratio for ranking, as the number of whispers in Cluster #4 containing this keyword divided by the total number of whispers with this word. We exclude common stopwords [9] and low-frequency words to avoid statistical outliers. A mere glance at Table VIII reveals that Cluster #4 users are focused on exchanging sexual content. Terms like “20f,” “f,”“17,” and “lesbian” indicate age, gender (f = female), and sexual orientation. Other frequently used words are associated with the exchange of nude photos (“trade,” “shower,” “nipples”) or more general erotic terms.

Users Who Get Blocked. Within Cluster #4, we find a subcluster of 1,412 users who often get blocked by others (Cluster #4-2-1 in Figure 32). As shown in Figure 33, these users have more “being-blocked” events in their clickstreams. In the meantime, as members of Cluster #4, these users are also highly likely to block other users.

Then the question is how often blocks are “bidirectional”—that is, user X blocks Y and then Y immediately blocks X. Unfortunately, our dataset cannot directly measure bidirectional blocks. For a blocking event, the known information includes the whisperID where two users chat, the userID issuing the block, but not the userID being blocked. Thus, we take an approximation approach to match potential “bidirectional” blocks (as an upper bound). For each user, we group his or her blocking and being-blocked events under the same whisperID as a *pair* if their time interval is within a short time window (e.g., 1 hour). This approximates immediate blocking back after getting a block. Figure 34 shows the matching result using the time window of 1 hour. Users in Cluster #4, particularly in Cluster #4-2-1, have a higher number of paired blocking events. It is likely these users are easily offended or often offend other users during private chat, suggesting a strong hostile behavior. We also test a 10-minute and 1-day time window and have similar conclusions.

10.3. Case Study 3: Renren Sybil Accounts

Finally, we analyze the Sybil cluster in Renren (Cluster #2 in Figure 25). Our system groups Sybil accounts into one single cluster with high accuracy. Based on the ground-truth labels in Renren data, we find 95% of true Sybils are clustered into the cluster, and only 0.74% of normal users are misclassified. The selected features indicate Sybils are more likely to engage in sending friend requests. This makes sense because a Sybil must first befriend a user before accessing private information or spamming.

Different Sybil Attack Strategies. Our system uncovers more fine-grained subclusters within the Sybil cluster, representing different attack strategies. Here we focus on the largest five (out of eight subclusters), which encompass 99.36% of Sybil accounts. Table IX shows their behavioral statistics. First, S_3 appears to describe “crawlers” who specialize in collecting user information and photos for sale on the black market [40]. Second, S_1, S_2, and S_4 all focus on “sending friend requests.” Sybils in S_1 send requests in bulk via Renren’s friend recommendation system, resulting in a high volume of

Table IX. Characteristics of Users in the 5 Biggest Sybil Clusters (S_1–S_5) and the Normal User Cluster (We Add the Cluster Label Based on the Selected Action Patterns per Cluster; “FrdReq” Stands for “Friend Requests”)

ID	Cluster Label	# Users	FriendReq per Day	ProfileReq per Day	FriendReq: In/Out
S_1	Friending in bulks	4,064	**25.13**	0.30	0.002
S_2	Friending quickly	1,891	19.81	2.08	0.004
S_3	Crawl profiles	1,348	11.41	**6.44**	0.050
S_4	Friending slowly	899	**8.76**	1.93	0.00004
S_5	Receive FrdReq	129	25.65	3.43	**0.286**
#1	Normal users	6,141	1.65	2.80	1.06

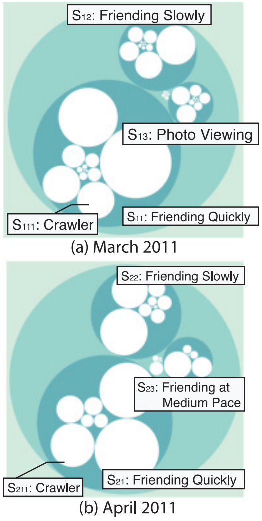


Fig. 35. Sybil clusters of two consecutive snapshots.

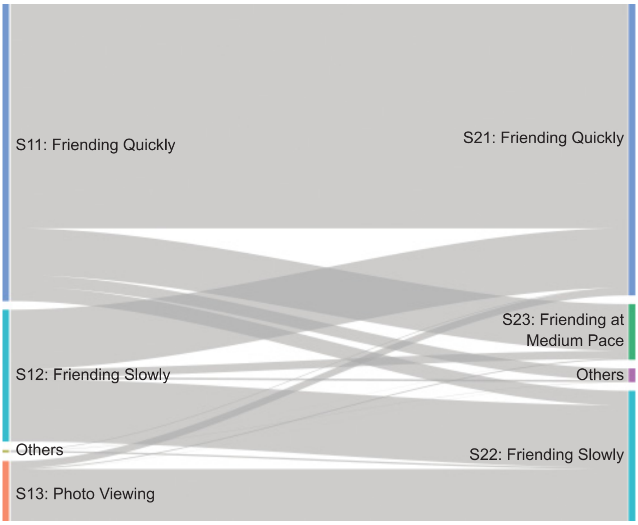


Fig. 36. Migration of Sybil accounts across snapshots.

friend requests per day (25.13). On the other hand, Sybils in S_4 tend to build social connections slowly (8.76 requests per day), possibly to avoid being detected. Finally, Sybils in S_5 are likely to *receive* friend requests. The ratio of incoming friend requests over outgoing ones is notably higher (0.286) than other Sybil clusters (<0.05). One possible explanation is that these Sybils are controlled by a single attacker to befriend each other to bootstrap their social connections.

Sybil Behavior Changes. Finally, we examine the behavior changes of Sybil accounts over our data collection period (i.e., 2 months). We divide Sybil clickstream data into two snapshots (one per month) and perform clustering on each snapshot. For cross-comparison purposes, we only consider Sybil accounts that appear in both snapshots (2,583 accounts). The results are displayed in Figure 35. We manually examine the top features in each cluster and add labels to the clusters.

We have three key observations: First, in the March snapshot, we find a cluster of Sybils that view photos in a similar way as normal users (S_{13}), but they then act differently in the April snapshot. We suspect (but cannot confirm) that these are legitimate accounts compromised by attackers [18]. Second, in the April snapshot, there is a new cluster where Sybils send friends requests at a medium speed (S_{23}). These Sybils often have a series of fast friend requests followed by a long pause. It is likely they are trying to maximize attack efficiency without triggering Renren’s limit on number of requests per day. Finally, our snapshot results do not include the original S_1 cluster in Figure 25 where Sybils send friending requests in bulk via Renren's recommendation system. This is because those Sybils were quickly banned in March and did not appear in the April snapshot. Also, the original group of crawlers (S_3) now is a subcluster at the secondary layer (S_{111} and S_{211}).

Figure 36 shows how Sybils migrate across two consecutive snapshots. Clearly, some users are reducing their friending request frequency possibly to avoid detection (S_{11} to S_{22}/S_{23}). On the other hand, some slow Sybils (S_{12}) also accelerate their friend requests in the next snapshot (S_{21}). In addition, we find that the majority of users who look like normal users in March (S_{13}) migrate to Sybil cluster S_{22}, where they send friend requests slowly. Also, it suggests that the attackers try to act stealthily by delivering friend requests in a slow speed instead of quickly sending bulks of requests immediately. In this way, attackers try to avoid alerting the account owners.

11. EVALUATION: CLUSTER QUALITY

Finally, we evaluate the quality of behavioral clusters produced by our system by examining how well they capture similar users. For this analysis, we compare our algorithm with existing clustering methods. In addition, we briefly discuss the system implementation and scalability tests.

11.1. Clustering Quality

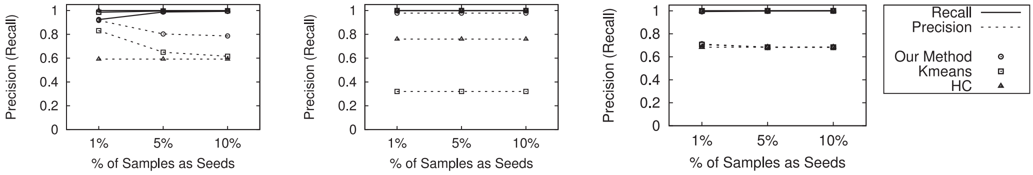
At the high level, an effective clustering algorithm should accurately group similar users together while separating different ones. We evaluate the quality of our behavioral clusters by testing how well they capture similar users. More specifically, given a small sample of known users, how accurately can they retrieve other users of the same type?

Experiment Setups. We first explain our experiment method, using Sybil detection in Renren as an example. Suppose a small sample of Sybils is known to us (x%). To detect the rest of the Sybil accounts, we use the known samples as *seeds* to color Renren's behavioral clusters. Any cluster that contains a known Sybil will be colored as Sybil-cluster (uncolored ones as normal). We evaluate the accuracy using two metrics: *Precision* (percentage of users in Sybil-clusters that are true Sybil accounts) and *Recall* (percentage of true Sybils that are captured by Sybil-clusters). A higher precision and recall indicate a better clustering quality. We vary the parameter x (1%, 5%, 10%) and repeat each experiment 10 times.

To perform this experiment on the Whisper dataset, we need to construct known groups of users. We use the two types of users identified in earlier analysis: *Dormant* users who have zero active events (16,688 users) and *Blocked* users who have been blocked at least once in a private chat (68,302 users).

Comparison Baselines. Our baselines are two widely used clustering algorithms: K-means algorithm [22] and Hierarchical Clustering (HC) algorithm [32]. We run both algorithms to cluster the full similarity graph (without feature pruning). At the high level, K-means divides users into K clusters where each user is assigned to the nearest cluster (center). The number of clusters K must be predefined. Here we generate multiple versions of K-means clusters and pick the K with the highest clustering quality (modularity). As a result, K-means generates 10 clusters on the Renren dataset and 10 for Whisper. In the same way, HC generates seven clusters for Whisper and two clusters for Renren.

Results. First, for Sybil detection on Renren, our algorithm is highly accurate with a precision of 93% and a recall of 94% (1% ground truth as seed) as shown in Figure 37(a).



(a) Sybils (Renren) (b) Dormant Users (Whisper) (c) Blocked Users (Whisper)

Fig. 37. The precision and recall of using the behavioral clusters to detect certain types of users. We compare our method with the K-means and Hierarchical Clustering (HC) algorithms.

Table X. Computation Time of Different Clustering Algorithms (the Computation Time Includes Two Major Parts: Distance Matrix Computation and Clustering)

	Whisper (100K Users)			**Renren (16K Users)**		
Algorithm	Total	Distance Matrix	Clustering	Total	Distance Matrix	Clustering
Our Method	58 hours	33 hours	25 hours	47 minutes	16 minutes	31 minutes
Simple HC	39 hours	33 hours	6 hours	18 minutes	16 minutes	9 minutes
K-means	49 hour	33 hours	16 hours	2 minutes	16 minutes	67 minutes

Using more seeds (e.g., 5%) produces a higher recall (99%) but reduces precision (82%). Nonetheless, the overall performance is better than K-means and HC (precision 67% and 61%). On the Whisper dataset, our algorithm achieves accurate results (98% precision, 100% recall) in identifying dormant users (Figure 37(b)). K-means and HC have a much lower precision (32% and 78%) with the same recall. Finally, all three algorithms achieve similar accuracy in detecting blocked users (73% precision and 99% recall). These results indicate that our system produces high-quality clusters to capture similar users.

12. IMPLEMENTATION AND SCALABILITY

Implementation. Our system is implemented in Python and runs on nine servers (HP DL360P Gen8). The clustering algorithm includes two major steps: (1) distance matrix computation, where we calculate the distance (or similarity) between any give node pairs, and (2) graph partitioning (or clustering) using iterative feature pruning algorithm. Note that the distance matrix computation can be parallelized, and we run this step using nine servers (four threads per server). The graph partitioning step can only be partially parallelized: we use a single thread to split all users into the first-level clusters (centralized); then, for any new subclusters, we recursively assign new threads to handle its splitting (parallelized).

Performance. We run our system on the Whisper dataset (100K users) and Renren dataset (16K users). We record the running time of each step in Table X. For the Whisper dataset, it takes about 58 hours to generate the complete behavioral clusters. For the smaller Renren data, it only takes 47 minutes. This performance is already sufficient for practical usage to build behavioral models on a weekly basis. Because these servers are a shared resource with other research teams, we only take four threads per server. Potentially, we can speed this up by increasing the server utility (e.g., 40 threads).

In addition, Table X shows our algorithm has comparable performance to simpler algorithms such as K-means and simple hierarchical clustering. For instance, it takes about 58 hours for our algorithm to cluster 100,000 Whisper users' clickstreams; it is only slightly quicker for K-means (49 hours) and simple HC (39 hours), but those algorithms don't produce natural cluster hierarchy or extract key features for cluster interpretation. Also, note that our algorithm is actually faster than K-means on the smaller Renren dataset.

To scale up the system to handle even larger datasets (e.g., 236 million users in Renren), one cannot simply add more machines. Despite the parallelization, the primary computational bottleneck is still in constructing the distance matrix for the similarity graph due to the high complexity. For a dataset of n users, the time complexity is $O(n^2)$. A promising approach is *incremental clustering*. The key idea is similar to the mechanism in Section 5: we take a small sample from a clickstream dataset to build the initial clusters (e.g., K clusters). Then we incrementally assign the rest of the users to existing clusters, based on their distance to the "centers" of existing clusters. In this way, we only need to compute each node's distance to the K cluster centers. The time complexity becomes $O(nK)$, where K is a small number. For Sybil detection, incremental clustering can already handle 1 million users in the real-world evaluation in Renren (Section 7). For unsupervised behavior modeling, there are open questions regarding how to perform sampling and how to maintain the consistency of the cluster hierarchy, which we leave to future work.

13. CONCLUSION & FUTURE WORK

In this work, we describe a clickstream analysis framework to model online user behavior and detect malicious user accounts. The resulting systems achieve accurate behavior detection and easy interpretation. For a given clickstream dataset, our model automatically identifies clusters of users with similar clickstream activities. Our results show that we can build an accurate Sybil detector by identifying and coloring clusters of "similar" clickstreams. To the best of our knowledge, this is the first work to leverage clickstream models for detecting malicious users in online social networks. Our system has been validated on ground-truth data, and a prototype has already detected new types of image-spam attacks on Renren and previous unknown Sybils on LinkedIn.

In addition, we extend this model beyond attacker classification to understanding more fine-grained user behaviors. The new model captures the natural hierarchical structure for user clusters. With a visualization tool, service providers can explore dominating user behaviors and categories as an overview, while tracking fine-grained user behavioral patterns along each category. Our tool does not require prior knowledge or assumptions of user categories (unsupervised), and thus, it can effectively capture unexpected or previously unknown behaviors. We demonstrate its effectiveness using case studies on real-world online social networks at Renren and Whisper.

We believe our proposed techniques are generalizable beyond online social networks. For example, Wikipedia, news, or Q&A sites might extract events based on the category or topic of the pages. E-commerce websites can define user events based on the functionality of the clickable links or product categories. Crowdsourcing sites can define click events based on the crowdsourcing workflow. In future work, we will explore broader applications of clickstream behavioral models and expand our tool to other user-driven systems.

REFERENCES

[1] Arindam Banerjee and Joydeep Ghosh. 2000. Concept-based clustering of clickstream data. In *Proceedings of the International Conference on Information Technology (ICIT'00)*.

[2] Arindam Banerjee and Joydeep Ghosh. 2001. Clickstream clustering using weighted longest common subsequences. In *Proceedings of the Web Mining Workshop at the 1st SIAM Conference on Data Mining (CDM'01)*.

[3] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. 2010. Detecting spammers on twitter. In *Proceedings of the Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference (CEAS'10)*.

[4] Fabricio Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgilio Almeida. 2009. Characterizing user behavior in online social networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'09)*.

[5] Alex Beutel. 2016. User behavior modeling with largescale graph analysis. *Ph.D. Thesis at Carnegie Mellon University*.

[6] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. 2013. CopyCatch: Stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the International Conference on World Wide Web (WWW'13)*.

[7] Erik W. Black, Kelsey Mezzina, and Lindsay A. Thompson. 2016. Anonymous social media understanding the content and context of Yik Yak. *Comput. Human Behav.* 57 (2016), 17–22.

[8] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment (JSTAT)* 10 (2008), P10008.

[9] Armand Brahaj. 2009. English Stop Words. Retrieved from http://xpo6.com/list-of-english-stop-words/.

[10] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. 2012. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*.

[11] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*.

[12] Rory Cellan. 2012. Facebook has more than 83 million illegitimate accounts. BBC News. (August 2012).

[13] Denzil Correa, Leandro Arajo Silva, Mainack Mondal, Fabrcio Benevenuto, and Krishna P. Gummadi. 2015. The many shades of anonymity: Characterizing anonymous social media content. In *Proceedings of the AAAI International Conference on Web and Social Media (ICWSM'15)*.

[14] George Danezis and Prateek Mittal. 2009. SybilInfer: Detecting sybil nodes using social networks. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'09)*.

[15] John R. Douceur. 2002. The Sybil attack. In *Proceedings of Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*.

[16] Martin Ester, Hans Peter Kriegel, Jrg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'96)*.

[17] Facebook. 2013. Verify Facebook Account. Retrieved from https://www.facebook.com/help/398085743567023/.

[18] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y. Zhao. 2010. Detecting and characterizing social spam campaigns. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'10)*.

[19] R. Stuart Geiger and Aaron Halfaker. 2013. Using edit sessions to measure participation in wikipedia. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'13)*.

[20] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. 2010. @spam: The underground on 140 characters or less. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'10)*.

[21] Şule Gündüz and M. Tamer Özsu. 2003. A web page prediction model based on click-stream tree representation of user behavior. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'03)*.

[22] John A. Hartigan and Manchek A. Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Appl. Stat.* 28 (1979), 100–108.

[23] Jeffrey Heer and Ed H. Chi. 2002a. Mining the structure of user activity using cluster stability. In *Proceedings of the Workshop on Web Analytics at SIAM Conference on Data Mining (CDM'02)*.

[24] Jeffrey Heer and Ed H. Chi. 2002b. Separating the swarm: Categorization methods for user sessions on the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*.

[25] Peter I. Hofgesang and Wojtek Kowalczyk. 2005. Analysing clickstream data: From anomaly detection to visitor profiling. In *Proceedings of the ECML/PKDD Discovery Challenge (DC'05)*.

[26] Homa Hosseinmardi, Richard Han, Qin Lv, Shivakant Mishra, and Amir Ghasemianlangroodi. 2014. Analyzing negative user behavior in a semi-anonymous social network. *CoRR* abs/1404.3839 (2014).

[27] Danesh Irani, Marco Balduzzi, Davide Balzarotti, Engin Kirda, and Calton Pu. 2011. Reverse social engineering attacks in online social networks. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'11)*.

[28] Jing Jiang, Christo Wilson, Xiao Wang, Peng Huang, Wenpeng Sha, Yafei Dai, and Ben Y. Zhao. 2010. Understanding latent interactions in online social networks. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'10)*.

[29] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2014. CatchSync: Catching synchronized behavior in large directed graphs. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD'14)*.

[30] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. 2016. Catching synchronized behaviors in large networks: A graph mining approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 4 (2016), 35:1–35:27.

[31] George Karypis and Vipin Kumar. 1998. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.* 48 (1998), 96–129.

[32] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis*. Vol. 344. John Wiley & Sons.

[33] Haewoon Kwak, Jeremy Blackburn, and Seungyeop Han. 2015. Exploring cyberbullying and other toxic behavior in team competition online games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'15)*.

[34] Michael Levandowsky and David Winter. 1971. Distance between sets. *Nature* 234 (1971), 34–35.

[35] Yixuan Li, Oscar Martinez, Xing Chen, Yi Li, and John E. Hopcroft. 2016. In a world that counts: Clustering and detecting fake social engagement at scale. In *Proceedings of the International Conference on World Wide Web (WWW'16)*.

[36] Lin Lu, Margaret Dunham, and Yu Meng. 2005. Mining significant usage patterns from clickstream data. In *Proceedings of the Web Mining and Web Usage Analysis Workshop (WebKDD'05)*.

[37] Justin Matejka, Tovi Grossman, and George Fitzmaurice. 2013. Patina: Dynamic heatmaps for visualizing application usage. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*.

[38] Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. 2010. Measuring the mixing time of social graphs. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'10)*.

[39] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2010. Re: CAPTCHAs—Understanding CAPTCHA-solving from an economic context. In *Proceedings of the USENIX Security Symposium (Usenix Security'10)*.

[40] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. 2011b. An analysis of underground forums. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'11)*.

[41] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. 2011a. Dirty jobs: The role of freelance labor in web service abuse. In *Proceedings of the USENIX Security Symposium (Usenix Security'11)*.

[42] Arjun Mukherjee, Bing Liu, and Natalie Glance. 2012. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the International Conference on World Wide Web (WWW'12)*.

[43] Jaimie Y. Park, Neil O'Hare, Rossano Schifanella, Alejandro Jaimes, and Chin-Wan Chung. 2015. A large-scale study of user image search behavior on the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'15)*.

[44] John C. Platt. 1999. Fast Training of Support Vector Machines Using Sequential Minimal Optimization. In *Advances in Kernel Methods*. MIT Press, 185–208.

[45] Narayanan Sadagopan and Jie Li. 2008. Characterizing typical and atypical user sessions in clickstreams. In *Proceedings of the International Conference on World Wide Web (WWW'08)*.

[46] Stan Salvador and Philip Chan. 2004. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *Proceedings of the Annual IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*.

[47] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. 2009. Understanding online social network usage from a network perspective. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'11)*.

[48] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang N. Tan. 2000. Web usage mining: Discovery and applications of usage patterns from Web data. *SIGKDD Explor. Newsl.* 1, 2 (2000), 12–23.

[49] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. 2010. Detecting spammers on social networks. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC'10)*.

[50] Qiang Su and Lu Chen. 2015. A method for discovering clusters of e-commerce interest patterns using click-stream data. *Electronic Commerce Research and Applications (ECRA)* 14, 1 (2015), 1–13.

[51] John R. Suler and Wende L. Phillips. 1998. The bad boys of cyberspace: Deviant behavior in a multimedia chat community. *Cyberpsy. Behav. Soc. Network.* 1, 3 (1998), 275–294.

[52] Kurt Thomas, Chris Grier, and Vern Paxson. 2012. Adapting social spam infrastructure for political censorship. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'12)*.

[53] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. 2011. Suspended accounts in retrospect: An analysis of twitter spam. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'11)*.

[54] I-Hsien Ting, Chris Kimble, and Daniel Kudenko. 2005. UBB mining: Finding unexpected browsing behaviour in clickstream data to improve a web site's design. In *Proceedings of the International Conference on Web Intelligence (ICWI'05)*.

[55] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. 2009. Sybil-resilient online content voting. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI'09)*.

[56] Cecilia Vega. 2012. Yelp Outs Companies That Pay for Positive Reviews. ABC News. Retrieved from http://abcnews.go.com/blogs/business/2012/11/yelp-outs-companies-that-pay-for-positive-reviews.

[57] Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. 2010. An analysis of social network-based sybil defenses. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'10)*.

[58] Alex Hai Wang. 2010. Don't follow me: Spam detection on twitter. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT'10)*.

[59] Gang Wang, Manish Mohanlal, Christo Wilson, Xiao Wang, Miriam Metzger, Haitao Zheng, and Ben Y. Zhao. 2013. Social turing tests: Crowdsourcing sybil detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'13)*.

[60] Gang Wang, Bolun Wang, Tianyi Wang, Ana Nika, Haitao Zheng, and Ben Y. Zhao. 2014. Whispers in the dark: Analysis of an anonymous social network. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'11)*.

[61] Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y. Zhao. 2012. Serf and turf: Crowdturfing for fun and profit. In *Proceedings of the International Conference on World Wide Web (WWW'12)*.

[62] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. 2006. Visualization of large hierarchical data by circle packing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*.

[63] Jishang Wei, Zeqian Shen, Neel Sundaresan, and Kwan-Liu Ma. 2012. Visual cluster exploration of web clickstream data. In *Proceedings of the IEEE Conference on Visual Analytics Science and Technology (VAST'12)*.

[64] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao. 2009. User interactions in social networks and their implications. In *Proceedings of the European Conference on Computer Systems (EuroSys'09)*.

[65] Yiming Yang and Jan O. Pedersen. 1997. A comparative study on feature selection in text categorization. In *Proceedings of the International Conference on Machine Learning (ICML'97)*.

[66] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. 2011. Uncovering social network sybils in the wild. In *Proceedings of the ACM SIGCOMM Conference on Internet Measurement (IMC'11)*.

[67] Sarita Yardi, Daniel Romero, Grant Schoenebeck, and Danah Boyd. 2010. Detecting spam in a Twitter network. *First Monday* 15, 1 (2010).

[68] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. 2008. SybilLimit: A near-optimal social network defense against Sybil attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (IEEE S&P'08)*.

[69] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham Flaxman. 2006. SybilGuard: Defending against Sybil attacks via social networks. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'06)*.

[70] Jian Zhao, Zhicheng Liu, Mira Dontcheva, Aaron Hertzmann, and Alan Wilson. 2015. MatrixWave: Visual comparison of event sequence data. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'15)*.

Received June 2016; revised January 2017; accepted March 2017