Eavesdropping on Fine-Grained User Activities Within Smartphone Apps Over Encrypted Network Traffic

Brendan Saltaformaggio[1], Hongjun Choi[1], Kristen Johnson[1], Yonghwi Kwon[1],
Qi Zhang[1], Xiangyu Zhang[1], Dongyan Xu[1], John Qian[2]

[1]*Department of Computer Science and CERIAS, Purdue University*
{*bsaltafo, choi293, john1187, kwon58, zhan2032, xyzhang, dxu*}*@cs.purdue.edu*

[2]*Cisco Systems*
johnq@cisco.com

Abstract

Smartphone apps have changed the way we interact with online services, but highly *specialized* apps come at a cost to privacy. In this paper we will demonstrate that a passive eavesdropper is capable of identifying *fine-grained user activities* within the wireless network traffic generated by apps. Despite the widespread use of fully encrypted communication, our technique, called NetScope, is based on the intuition that the highly specific implementation of each app leaves a fingerprint on its traffic behavior (e.g., transfer rates, packet exchanges, and data movement). By learning the subtle traffic behavioral differences between activities (e.g., "browsing" versus "chatting" in a dating app), NetScope is able to perform robust inference of users' activities, *for both Android and iOS devices*, based solely on inspecting IP headers. Our evaluation with 35 widely popular app activities (ranging from social networking and dating to personal health and presidential campaigns) shows that NetScope yields high detection accuracy (78.04% precision and 76.04% recall on average).

1 Introduction

Smartphone apps have replaced web browsers for interacting with many online services (e.g., media streaming, social networking, lifestyle, and finances) [23]. However, these highly specialized apps leave behind distinct traces of their activities in wireless network traffic. In this paper, we will demonstrate that a passive eavesdropper is capable of identifying *fine-grained user activities* within apps, despite the use of traffic encryption, *based solely* on inspecting IP packet headers and metadata. This capability highlights new challenges in security and privacy: For example, the inference of a user's in-app activities can reveal highly sensitive information based on the nature of many apps, such as those for adult dating (e.g., frequently browsing versus chatting with matches on the Ashley Madison app) or personal health (e.g., looking up nearby HIV clinics).

Prior efforts in smartphone traffic analysis have developed tools for finger-printing mobile phones or individual apps [7, 11, 29, 44, 46] or modeling smartphone usage behavior [14, 27, 31, 36, 43]. These often require prior physical access to the devices of interest [14] and many only perform traffic-content signature matching [11, 27, 44, 46] or protocol identification [36]. However, nearly all apps today make use of (encrypted) SSL/TLS communication, and thus packet content analysis (e.g. DPI) and protocol identification reveal little information about smartphone apps. More importantly, because of user mobility, apps may only perform *a portion* of their communication while connected to any single Wi-Fi network — giving eavesdroppers only a small window of (encrypted) traffic to inspect. In fact, many existing solutions [7,11,29,43,44,46] will miss an app if its signature does not occur in that window. We call this the *transient connectivity* challenge.

In this paper, we overcome these challenges and show that even a small window of encrypted traffic can reveal an app's semantic activities. Intuitively, an app's highly customized implementation generates distinctive traffic patterns (e.g., transfer rates, packet exchanges, and data movement) for each of that app's activities. We call this the activity's *traffic behavior*. For example, the Facebook app exhibits a much different traffic behavior while the user is reading posts versus posting a new status update, which differs further from the traffic behavior of tweeting via the Twitter app. By leveraging *traffic behavioral clues*, we can achieve fine-grained monitoring of a user's actions, *without inspecting the packets' contents*.

We present NetScope, a technique that utilizes traffic behavioral clues to automatically build a detector for smartphone (*both* Android and iOS) app activities. The use of NetScope is intuitive: First, an eavesdropper performs offline training with the apps of interest, during which NetScope automatically builds models of the apps' human-observed, semantic activities from the measured traffic behaviors. NetScope requires no packet

content and no access to/knowledge of any target (victim) devices. The traffic measurements are converted to feature sets, and a *behavioral feature clustering* method is used to isolate similar behaviors. The most distinct behaviors are learned by *two complementary machine learning models* which NetScope packages into a *detection module* to be deployed at Wi-Fi access points (or other network traffic collection devices) for lightweight, online monitoring of users' activities.

We have evaluated NetScope in a lab deployment involving 7 different users with 2 iPhones and 5 different Android phones. The 35 subject app activities range from generic apps (e.g., Facebook, YouTube) to highly specialized apps for dating (e.g., OkCupid, Ashley Madison), health (e.g., HIV monitoring), and political campaigns (those of Bernie Sanders and Ben Carson). NetScope is shown to detect this variety of activities with high accuracy. To the best of our knowledge, NetScope is among the first to enable smartphone app activity eavesdropping from IP headers only and, by doing so, reveal new privacy implications of using specialized, privacy sensitive apps via public/monitored Wi-Fi networks.

2 Challenges and Solution Overview

Traditional (non-smartphone) traffic analysis techniques rely on deep packet inspection (DPI) [10,16,28], protocol identification [6, 20, 34, 35] and, more recently, fingerprinting encrypted website-traffic [4, 15, 21, 24, 33] and detecting protocols post-encryption [39, 42]. Unfortunately, recent studies [11,31,43,44] have shown that the new paradigms of mobile app network communication limit their applicability.

For privacy, apps direct all traffic through SSL/TLS connections. Hence traffic signatures and DPI cannot be applied to the majority of mobile apps, and identifying specific *values* within an app's traffic is impossible. Further, as observed in [31,43], apps' traffic follows vastly different patterns (e.g., persistent, with both server and client tracking semantic context and state) than that of HTTPS traffic, so encrypted web-traffic fingerprinting techniques [4,15,33] would be unable to interpret an app's traffic. Still, some post-encryption protocol detection tools could apply to app protocols [35,39,42], thus we are motivated to enable much more *fine-grained* detection of semantic user-actions within apps' traffic.

Prior work has assumed that apps may be identified by the domain name or IP addresses with which they communicate [7, 36]. However, this simple heuristic is too coarse-grained and error-prone. Many services are hosted in commercial clouds (e.g., many of our test cases only use Microsoft Azure). Moreover, cloud-hosted services make use of load balancers, making it impossible to map back-end server's IP addresses to services. Most importantly, NetScope's goal is *not* just to identify the app, but to identify actions within the app, which is impossible via only IP/hostname resolution.

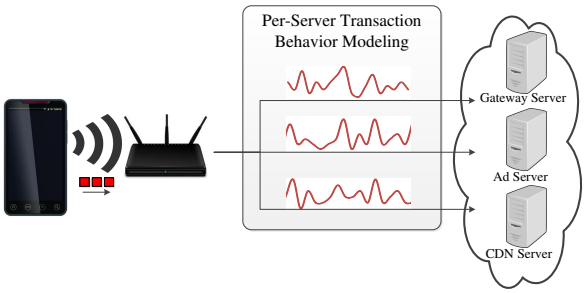

Figure 1: NetScope models each server transaction's fine-grained traffic behavior separately.

Further, mobile apps may *only perform a portion of their network communication over any single wireless network*. This is because smartphones may switch between the cellular network and in-range Wi-Fi networks seamlessly. In the past, network communications were modeled as automata with state transitions based on traffic patterns [6]. However, this is no longer effective for apps, because a single network may only observe a subset of an app's traffic (missing the beginning, end, or both). Essentially, the eavesdropper "drops in" on the middle of the app's communication. We call this the *transient connectivity* challenge. To the best of our knowledge, no existing traffic analysis tools consider (nor overcome) this challenge.

2.1 Traffic Behavior-Based Inference

In light of the above challenges, we propose modeling the traffic's *fine-grained behaviors*. Intuitively, each app is implemented differently, as determined by its semantics. We observe that such implementation differences induce *activity-specific traffic behaviors*. Each activity within an app (e.g., posting to versus reading Facebook) will generate discernible differences in their traffic. NetScope leverages such differences to infer the user's activities using *app behavior-based traffic models*. To build these models, NetScope decomposes the traffic based on several design features of modern apps' network connections. As a running example, Figure 1 shows how NetScope observes an app's network behavior from traffic handled by a Wi-Fi access point.

During our app profiling, we observed that each activity will connect to multiple servers in parallel, each with a specific purpose. For simplicity, Figure 1 shows one activity connecting to 3 servers: gateway, ad, and CDN servers. The app's communication with each server behaves differently based on that server's purpose. NetScope leverages this per-server behavior to build models for each *server transaction* (a packet stream between the device and a remote server) independently, as shown in Figure 1. In Figure 1 each server transaction's behavior is simplified as a curve, but in fact,

NetScope uses 26 metrics per measurement to model the traffic's behavior (Section 3).

During offline training, NetScope makes fine-grained (5 ms) measurements of each server transaction's behavior. These measurements are grouped based on their similarity, and the most unique behaviors are chosen to build models. Intuitively, NetScope aims to detect the server transactions by matching models of their most distinctive behaviors. Because the measurements are taken over such small time intervals, each behavior model represents a fine-grained (sub-second) portion of a transaction's traffic. During online detection, this allows NetScope to overcome the *transient connectivity* challenge because each model can match only a small subset of the app's traffic.

3 NetScope Design

NetScope's training is performed offline once to build traffic behavior models. To collect training data, NetScope will listen on a training Wi-Fi access point (to which only the training device is connected) and log the IP headers and relative timestamps for all packets it observes. Training is conducted such that the user (eavesdropper) only needs to perform a semantic activity (e.g., viewing a YouTube video) on the training device and then give NetScope the name of the activity that is being modeled (e.g., "YouTube play"). Note also that such training could be more automatic via existing smartphone UI exercising utilities [1, 2].

3.1 Feature Extraction

A behavior model is a representation of how that traffic "moves" through the access point *per server transaction*. NetScope partitions the traffic log into server transactions containing all the IP headers (in temporal order) which the device sent to/received from each remote server. Once the traffic log is partitioned into server send/receive transactions, each transaction's behavior will be modeled separately. However, because of the transient connectivity challenge, we cannot assume that the entire transaction will be observed during online detection. Thus, instead of computing the behavior of the entire transaction, we divide it into *behavior measurements* — a measurement of the traffic's behavior over a very small time window (5 ms in our implementation).

Lastly, care is taken when choosing metrics for the behavior measurements. In the next section, these metrics will be the feature sets for NetScope's machine learning algorithms, and thus they must be comparable between any observed network traffic. For example, packet counts would be misleading because the same activity may transmit data of variable sizes (e.g., long versus short text messages). Thus we designed the following metrics (26 data points total) to measure the traffic's *implicit behaviors* that are not *explicitly* observable from any packet content. Each of the following metrics is computed for every behavior measurement (i.e., 5 ms time interval) within the server transactions.

Send and Receive Average Inter-Packet Times are two measurements which consider how quickly the server is sending packets to the app and vice versa. NetScope computes an average of the time differences between every consecutive packet in the send and receive transactions:

$$AvgIPT(P) = \frac{\sum_{i=1}^{|P|-1}(ts_{i+1} - ts_i)}{|P|-1} \quad (1)$$

where P is the set of sent or received packets and ts_i is the timestamp of the i^{th} packet. Intuitively, these measure the transaction's "bursts" (i.e., many adjacent packets versus few distant packets).

Send and Receive Packet Count Ratios measure the ratios of the total number of packets that the app sends to and receives from the server (i.e., how "chatty" the server and app are). Specifically, if the app sends c_{send} packets to the server and receives c_{recv} packets then the send and receive ratios are calculated as:

$$PCR_{send} = \frac{c_{send}}{c_{send} + c_{recv}}, PCR_{recv} = \frac{c_{recv}}{c_{send} + c_{recv}} \quad (2)$$

Note that ratios, rather than the raw counts, allow the measurement to be generalizable to all activities we model.

Send and Receive Data Size Ratios measure the ratios of the total data sent to and received from the server. Unlike the transaction's "chattiness", this measurement represents how much data is flowing between the two. For an app which sends m_{send} bytes of data and receives m_{recv} bytes, the ratios are calculated as:

$$DSR_{send} = \frac{m_{send}}{m_{send} + m_{recv}}, DSR_{recv} = \frac{m_{recv}}{m_{send} + m_{recv}} \quad (3)$$

Comparing these metrics to the previous two reveals a significant amount about the activity, e.g., streaming video from the server (low chat, heavy download) versus instant messaging (high chat, similar data ratios).

Packet Size Classification considers the distribution of data sizes. For example, a transaction with 3 1024-byte packets should be modeled differently than 12 256-byte packets. To capture this relation, we divided the possible single packet size range into 10 disjoint ranges. NetScope computes the number of packets sent and received within the measurement window with sizes within each range and then normalizes by the total number of sent or received packets, respectively. The resulting 20 ratios model how the data is distributed among the packets being sent and received.

3.2 Building Behavioral Models

The majority of the behavior measurements will be similar across multiple activities, but each activity also contains enough unique behaviors to be distinguishable. Iso-

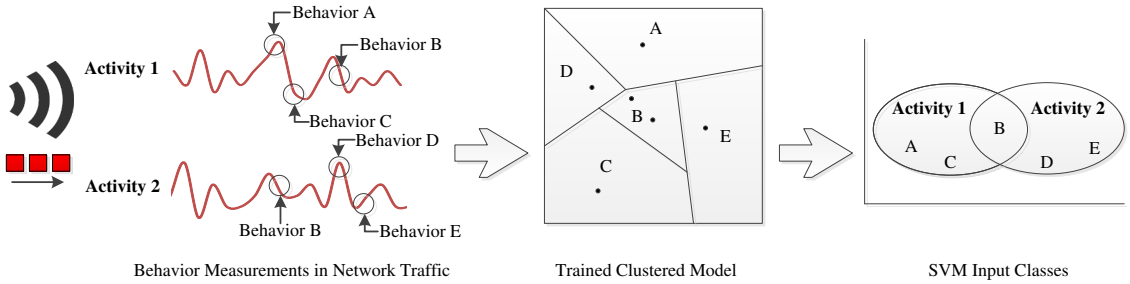

Figure 2: NetScope models each activity's unique traffic behaviors with two complementary machine learning models.

lating these unique behaviors within the behavior measurements (tens of thousands of them) can be modeled as a data mining/clustering problem. NetScope uses the K-means clustering (unsupervised machine learning) algorithm to partition the input feature sets into K clusters based on their distance from each other and the clusters' centers. The resulting clusters contain disjoint subsets of the behavior measurements. Among these, some are tightly clustered (the measurements within are highly similar) and some are loosely clustered (only somewhat similar to each other, but less similar to the other clusters). In this way, the clusters reveal which behavior measurements are most distinctive.

The K-means algorithm requires K (the number of clusters) as input. However, NetScope cannot know *a priori* how many clusters will be needed to identify the unique feature sets (a common problem in complex data mining applications). Thus NetScope relies on *an incremental clustering algorithm* to find an appropriate number of clusters based on the observed accuracy. Given N feature sets, NetScope runs the K-means clustering algorithm starting at $K = \frac{N}{8}$ until $K = \frac{N}{4}$ in 100 iterations. Each run yields new clusters and an average *inertia*, which measures how tightly the feature sets are clustered (the within-cluster sum of squares). NetScope plots the inertia values and performs a change point analysis to determine if the increasing K value is no longer yielding significant inertia improvements. If a change point is never detected, NetScope stops iterating at $K = \frac{N}{4}$ which, through experimentation, we find to be a reasonable cutoff (i.e., 1 cluster for every 4 feature sets). When a suitable K value is found, NetScope records which cluster each feature set belongs to. We call these clusters of behavior measurements *traffic behavior models*.

Figure 2 shows a simplified example: Two activities with one server transaction each. The network traffic yields 6 behavioral measurements, and of those, 4 are unique (both activities exhibit behavior B). Clustering these behavior measurements with $K = 5$ isolates the 4 unique behaviors. Note that behavior models are not activity or server transaction specific. As Figure 2 shows, they are derived from behavior measurements taken from every training activity. This makes behavior models more general (similar behaviors exhibited by different apps will share the same model) and more accurate compared to training separate cluster models. Intuitively, if NetScope can compare and contrast more behaviors, then each behavior model will be more precise.

NetScope must connect the generic behavior models to the activity(ies) which exhibit them. However, because of transient connectivity, NetScope cannot assume that it will later observe the exact same set of behaviors modeled. Thus, the models of each activity must allow for some behaviors to be missing during detection. Further, we want to use a model which will give more weight to unique behaviors and less weight to behaviors which are common among multiple activities. In Figure 2, the behavior models for A and C are stronger indicators of Activity 1, compared to the behavior model for B.

NetScope uses a multi-class support vector machine (SVM) — a widely used supervised machine learning method focusing on high classification accuracy with many possible observations. NetScope constructs a feature matrix: each row consisting of the app activity labels and binary features (i.e., values 0 or 1) representing if any server transaction within this activity produced the given behavior model. For the K identified behavior models, the matrix columns 2 through K+1 are each assigned 1 or 0 to represent if that activity produced that behavior model. The trained SVM model captures the relation that: given a set of behavior models, the model determines which activity produces a similar set of behaviors.

Notice that we did not explicitly separate the behavior models by server transaction, because the multi-class SVM will *implicitly* capture this relationship. Consider Figure 2: For Activity 1 assume that behaviors A and C are exhibited by two separate server transactions. The multi-class SVM model for Activity 1 will be trained with behaviors A, B, and C marked 1. As desired, this captures the relationship that detecting behaviors A and C lead to a more confident match of App 1 than just detecting A or C alone.

NetScope packages the cluster model and SVM into a *detection module* to be distributed at Wi-Fi access points to monitor smartphones for the trained behaviors.

3.3 User Activity Detection

The NetScope detection module takes as input a stream of IP packet headers and outputs labels for which activity behaviors it observes in the traffic. NetScope inspects traffic from different phones separately. For each packet that the detection module processes, it builds a set of server transactions. If the packet belongs to an on-going server transaction, then NetScope updates that transaction's behavior measurements. Otherwise, a new server transaction is registered and NetScope waits to collect enough packets for the first behavior measurement to be computed (as in Section 3.1). NetScope then determines which behavior model matches each new behavior measurement. To do this, NetScope consults the trained cluster model: Given an unknown behavior measurement, the cluster model will report which cluster the new behavior would fall in. At this point, NetScope does not consider if this measurement is *not* a known behavior (i.e., traffic which we did not train for), instead this will be handled naturally by the multi-class SVM model later.

Finally, for each new behavior model in the traffic, NetScope attempts to match the known set of concurrent behaviors with an activity's model. For this, NetScope builds a test feature set from the observed behavior models, and this unlabeled row is tested with the multi-class SVM model. This yields a list of probabilities representing how well each training matrix row matches the testing data. If no row matches above 60% then NetScope discards the result and continues collecting traffic. We chose a cutoff of 60% because we find that true matches occur with above 85% confidence, but mismatches (i.e. traffic we did not train for) result in less than 50% confidence. If any rows match above the cutoff, then NetScope reports the best matching row's label as a detection.

4 Evaluation

We have implemented NetScope in Python (~7K lines of code). Data collection is performed using the `tcpdump` utility and processed via the `dpkt` library [12]. For NetScope's machine learning models we employ the widely-used `scikit-learn` library [25].

4.1 Training and Deploying NetScope

We conducted the training as described in Section 3 using one Samsung Galaxy S4 phone and a set of 35 different user activities which we aimed to detect, summarized in Table 1. We chose the apps in Table 1 based on two criteria: 1) their top ranking among free apps in *both* the Google Play Store and Apple App Store or 2) their highly specialized, privacy sensitive activities. We repeated data collection for each activity 4 times. This processing took 52 minutes from start (input all collected training data) to finish (output trained NetScope detection module).

Deployment Setup To recreate a typical rogue Wi-Fi hotspot scenario, we set up a new Wi-Fi access point in our lab (for members of this project only). We installed packet header logging functionality on the access point and set this as the default Wi-Fi network for the 7 project members' personal smartphones: an HTC Desire 500, LG G2, LG G3, and 2 Samsung Galaxy S4s, as well as 1 iPhone 6 and 1 iPhone 6 Plus. Each project member interacted with the test apps to generate a variety of activities (along with any typical background traffic generated by each device) in the network trace for evaluation. In addition, four members' laptops solely used the Wi-Fi over the deployment period, but as Section 3.3 mentioned, NetScope ignored these devices during its operation. After collecting a sufficiently large Wi-Fi trace (containing a total of 667 app activities to detect), we replayed the entire trace to the NetScope detection module (as if the packets were arriving in real time) and recorded its detection results.

4.2 Detection Results

To obtain ground truth (i.e., the activities that were actually performed), all project members logged the date and time they performed any of the 35 activities from Table 1 (this was done via a script added to the users' smartphones). We processed these logs and NetScope's output to measure the detection module's accuracy.

Table 2 presents NetScope's detection results across all 7 smartphones. Columns 1, 2, and 3 show the activity, ground truth (number of times that the users performed that activity), and the number of times NetScope correctly detected that activity, i.e., the true positives (TP), respectively. Column 4 shows the number of times NetScope misclassified that activity as a different activity, and the times NetScope did not detect that activity occurring (not misclassified) is shown in Column 5. The sum of Columns 4 and 5 is the false negative count (FN). Column 6 shows the false positive (FP) count (other activities classified as that row's activity). Precision and recall are shown in Columns 7 and 8, respectively.

Table 2 shows that NetScope achieves very high detection accuracy. Column 7 shows that NetScope's average precision is 78.04%. This denotes that among all of NetScope's identifications, 78.04% of them are correct. Average recall is also high: 76.04%. This can be understood as 76.04% of the activity instances in the network traffic were correctly detected.

Table 2 shows that NetScope is sensitive enough to accurately distinguish between similar activities in different apps. For example, listening to music on the Pandora and Spotify apps both have precision above 76% and recall above 72%. From Table 2 we can see that even these similar activities provide distinguishing characteristics in their network behaviors.

Category	App	User Activity (Detection Target)	Training Label	Dominant Network Behavior
News & Politics	CNN News	Browse and read news articles	CNN Read	Download content, bursty
	Bernie Sanders 2016	Read stances and news updates	Sanders Read	
	Ben Carson 2016	Read stances and news updates	Carson Read	
Personal Health	HIV Atlas	Lookup treatment information	HIV Info	
		Lookup HIV test clinics	HIV Clinics	
Social	Facebook	Read Facebook Feed	Facebook Feed	
		Post to Facebook	Facebook Post	Upload content
	Twitter	Post new tweet	Twitter Tweet	
		Read tweets	Twitter Read	Download content, steady
	Instagram	Browse Posts	Instagram Browse	Download content, bursty
		Post to Instagram	Instagram Post	Upload content
	Snapchat	Photo Chat on Snapchat	Snapchat Chat	Interactive (bursty upload and download)
Dating	Tinder	Browse potential matches	Tinder Browse	
		Chat with connections	Tinder Chat	
	OkCupid	Browse potential matches	OkCupid Browse	
		Chat with connections	OkCupid Chat	
	Ashley Madison[1]	Browse potential matches	Ashley Madison Browse	
Travel & Local	Google Maps	Search location and view maps	Google Maps	
	Yelp	Browse Yelp	Yelp Browse	Download content, steady
		Search Yelp	Yelp Search	Download content, bursty
Shopping	Amazon	Browse online store	Amazon Browse	
Communication	Facebook Messenger	Chat with friends	Messenger Chat	Interactive
	Skype	Video call with friend	Skype Video	
		Voice call with friend	Skype Voice	
		Message chat with friend	Skype Chat	
	Gmail	Read email	Gmail Read	Download content, bursty
		Send email	Gmail Send	Upload content
	WhatsApp	Message chat with friend	WhatsApp Chat	Interactive
Media Streaming	Spotify Music	Navigate through playlists	Spotify Navigate	
		Listen to music	Spotify Listen	Download content, steady
	YouTube	Watch videos	YouTube Play	
		Search and browse videos	YouTube Navigate	Interactive
	Netflix	Browse through videos	Netflix Browse	Download content, bursty
		Watch Videos	Netflix Watch	Download content, steady
	Pandora	Listen to music	Pandora Listen	

Table 1: Training Activities for Various Apps with Diverse Network Behaviors.

Detection Time As an online eavesdropping tool, it is important that the detection module be light-weight and efficient in order to produce near real-time results. On average the classifier took 0.62 seconds to produce a result from input behavior measurements. Thus, any bottleneck for detection comes from collecting behavior measurements to match a behavior model. Through measurements performed during online deployment, we found that it took between 50 and 300 behavior measurements to match the activity models reliably. Thus it took between 0.25 seconds to 1.5 seconds of traffic observation to yield a result.

4.3 Device/Platform-Generic Detection

Surprisingly, these results were obtained via the NetScope detection module *trained with only one Samsung Galaxy S4*: A NetScope detection module is found to be *cross-platform*, working with both iPhone and Android traffic. To get a better idea of how NetScope performed with each device, we calculated the per device detection results.

Table 3 shows that, although being trained with only the Samsung Galaxy S4, NetScope performs well across all devices and platforms. Interestingly, we notice a "step-wise" effect in precision as the device's operating systems differ more from the training device's. Both Android 4.4.2 devices perform the best: the LG G3's precision is 89.6% and the training Samsung Galaxy S4's is 93.2%. The most closely related OS is Android 5.0, which also shows good results: the LG G2's precision is 74.29% and the (not training) Samsung Galaxy S4's is 74.07%. The remaining (most different) cases exhibit precisions between 43.43% and 72.04%. Note also that we *purposely* trained NetScope with a very restrictive data set (only 1 device and 4 repetitions of each activity) to evaluate the power of generalizing its signatures. In a real-world deployment, it would make more sense to train with data from each platform (or at least a variety of platforms) which one intends to observe during detection.

4.4 User Privacy Implications

NetScope's high detection accuracy raises serious privacy implications. While we by no means condone such applications, NetScope can be used to infer user privacy-sensitive information, especially from highly specialized personal apps.

To highlight this privacy impact, we have included HIV Atlas (one of the most popular HIV management apps) in our test cases. We tested NetScope with the

[1]The Ashley Madison app charges users for sending chat messages, so we excluded that activity from our training data set.

App Activity	Ground Truth	TP	Misclassify	Miss	FP	Precision	Recall
CNN Read	33	19	14	0	10	65.52%	57.58%
Sanders Read	25	25	0	0	1	96.15%	100.00%
Carson Read	21	13	8	0	2	86.67%	61.90%
HIV Info	24	13	8	3	0	100.00%	54.17%
HIV Clinics	24	19	5	0	8	70.37%	79.17%
Facebook Feed	36	15	21	0	20	42.86%	41.67%
Facebook Post	24	16	6	2	11	59.26%	66.67%
Twitter Tweet	24	17	7	0	2	89.47%	70.83%
Twitter Read	10	10	0	0	2	83.33%	100.00%
Instagram Browse	11	6	5	0	15	28.57%	54.55%
Instagram Post	11	4	7	0	0	100.00%	36.36%
Snapchat Chat	11	8	3	0	9	47.06%	72.73%
Tinder Browse	27	22	4	1	0	100.00%	81.48%
Tinder Chat	25	20	4	1	1	95.24%	80.00%
OkCupid Browse	19	17	2	0	7	70.83%	89.47%
OkCupid Chat	21	20	1	0	0	100.00%	95.24%
Ashley Madison Browse	22	21	1	0	1	95.45%	95.45%
Google Maps	34	34	0	0	3	91.89%	100.00%
Yelp Browse	11	8	3	0	5	61.54%	72.73%
Yelp Search	11	5	6	0	10	33.33%	45.45%
Amazon Browse	11	4	7	0	1	80.00%	36.36%
Messenger Chat	19	17	2	0	14	54.84%	89.47%
Skype Video	9	9	0	0	0	100.00%	100.00%
Skype Voice	9	9	0	0	0	100.00%	100.00%
Skype Chat	9	9	0	0	8	52.94%	100.00%
Gmail Read	11	11	0	0	5	68.75%	100.00%
Gmail Send	11	11	0	0	0	100.00%	100.00%
WhatsApp Chat	11	11	0	0	6	64.71%	100.00%
Spotify Navigate	18	18	0	0	2	90.00%	100.00%
Spotify Listen	16	13	3	0	4	76.47%	81.25%
YouTube Play	44	16	26	2	2	88.89%	36.36%
YouTube Navigate	42	30	12	0	14	68.18%	71.43%
Netflix Browse	11	4	7	0	0	100.00%	36.36%
Netflix Watch	11	9	2	0	4	69.23%	81.82%
Pandora Listen	11	8	3	0	0	100.00%	72.73%

Table 2: App Activity Detection Results.

Device	OS Version	Ground Truth	TP	Misclassify	Miss	FP	Precision	Recall
LG G3	Android 4.4.2	125	112	13	0	13	89.6%	89.6%
LG G2	Android 5.0	35	26	9	0	9	74.29%	74.29%
HTC Desire 500	Android 4.1.2	95	67	26	2	26	72.04%	70.53%
Samsung Galaxy S4	Android 5.0	88	60	21	7	21	74.07%	68.18%
Samsung Galaxy S4 (training)	Android 4.4.2	147	137	10	0	10	93.2%	93.2%
iPhone 6	iOS 8	78	46	32	0	32	58.97%	58.97%
iPhone 6 Plus	iOS 8	99	43	56	0	56	43.43%	43.43%

Table 3: App Activity Detection Results Calculated Per-Device.

two dominant features of HIV Atlas: looking up treatment information and looking up nearby HIV test clinics (Rows 4–5 in Table 2). NetScope's ability to recognize individual in-app activities is critical here: Identifying a person reading general HIV information is far less probative than monitoring someone searching for nearby HIV test clinics. Now, consider a malicious user connecting to the same Wi-Fi and sniffing all the IP packets. By correlating the inferred app activities with device type/name, connection times, and even visual observations, the eavesdropper could easily identify the individual who performed the HIV app activities.

Beyond targeted eavesdropping, NetScope's detection capability might be abused for broader violations of privacy. For example, given the recent studies linking the degree of casual dating app use and the spread of sexually transmitted diseases (STDs) [3, 26], authorities might consider secretly tracking how *actively* community members use these apps (e.g., passively browsing potential matches versus frequently chatting with their connections). Users are unlikely to agree to such monitoring. Table 2 shows that Tinder, OkCupid, and Ashley Madison (possibly the activity that users *most* want to keep secret) all have high detection accuracies with an average of 92.3% precision and 88.33% recall among all 5 of these apps' activities.

Another concerning scenario, is employee discrimination on the basis of political affiliation (which *is legal* in most states) [38]. The use of highly specialized apps, such as the Bernie Sanders and Ben Carson presidential campaign apps (Rows 2–3 in Table 2), reveal such political affiliations. These cases have reasonably high de-

tection accuracies: The Bernie Sanders app has only 1 false positive result yielding 96.15% precision and 100% recall; and the Ben Carson app has only 8 misclassifications yielding 86.67% precision and 61.9% recall.

5 Discussion

Imitation Attacks Like all statistical learning methods, NetScope can be vulnerable to imitation attacks — an attacker may invest a significant amount of effort to "replay" the exact traffic sent between a benign app and the servers it connects to. If the imitation was nearly identical to the original benign app, then NetScope may classify that device as performing the imitated app activities when in fact the user was not.

Traffic Obfuscation Defense To mitigate the privacy impacts highlighted in Section 4.4, developers may wish to degrade NetScope's effectiveness by adding *randomness* to an app's traffic behavior. This would require non-trivial changes to the app and servers involved. Obfuscated behaviors need to be generated for *each run* of that app to prevent NetScope from approximating the traffic during training. Because NetScope's models are server transaction specific, an app would need to obfuscate its traffic behavior for multiple servers that it connects to — incurring additional computation and network traffic overheads.

6 Related Work

Analysis of Encrypted Network Traffic Encrypted traffic has been the target of network analysis research for some time. A primary goal of this field of research has been fingerprinting website visits in encrypted traffic [4, 15, 17, 19, 21, 24, 30, 33, 42]. Several of these works have employed statistical analysis [4,30,39], naïve Bayes classifiers [17,19,22], and machine learning techniques [18,24]. Further, a recent study by Dyer et al. [13] found that traffic analysis countermeasures were still insufficient to prevent eavesdropping. Besides website fingerprinting there are many other works which analyze encrypted network traffic to uncover numerous other information leakages. One notable direction is the detection of languages, spoken words, or phrases in encrypted VoIP traffic [37,40,41].

NetScope shares a goal with these works: expose an information leak in secure communication. However, as discussed in Section 2, the design and usage of smartphones introduces a number of new challenges to mobile app traffic analysis.

Schneider et al. [28] extracted click-streams from passively monitored network traffic to identify user activities on social network sites. NetScope is also a passive network analysis tool which aims to detect user's activities, but the detection of website-based activities differs significantly from in-app user activities. Later, Verde et al. [32] proposed features which could track users behind a NAT. NetScope is similar to this work in that they both build and detect fingerprints from network flows, but NetScope aims for a more fine-grained identification (user's in-app activities). Also of note, Chen et al. [5] found a number of side-channel leakages in web-applications via traffic analysis which disclose sensitive information about its users.

Zhang et al. [45] proposed identifying coarse-grained user activities (e.g., web browsing, chatting, online gaming) via passive monitoring of 802.11 wireless traffic from laptops. Both this work and NetScope share similar adversary models and techniques, but NetScope's detection is significantly more fine-grained: detecting specific apps and activities. As discussed in Section 2, this work could hardly be ported to handle the challenges inherent in smartphone traffic analysis.

Smartphone Network Traffic Analysis To understand how smartphones were being used, many works aimed to model phone usage behavior [9, 27, 31, 43]. Among other features, ProfileDroid [36] analyzed network traffic to model an Android's usage. Falaki et al. [14] looked at how traffic patterns affect a smartphone's execution. Xu et al. [43] performed a large-scale investigation of apps' network usage and traffic invariants. Tongaonkar et al. [31] modeled app usage by tracking identifiers in ad libraries through traffic analysis. Building from these ideas, MAPPER [27] enforces per-app/user policies based on observed traffic patterns. Unfortunately, these works either rely on analysis of unencrypted network traffic [27, 31, 43], protocol identification [36], or on-device monitoring tools [14]; making these solutions poorly suited for spying on user's activities.

Networkprofiler [11] followed by FLOWR [44] automatically build traffic signatures of apps' unencrypted network communications. Unfortunately, modern apps use encrypted communication. Stöber et al. [29] aimed to identify the apps installed on an Android device by monitoring that device's network usage. NetScope builds from this idea to leverage many more network traffic features for much more fine-grained detection.

Most recently, Conti et al. [7, 8] found that the Facebook, Gmail, and Twitter apps produce different network patterns for several in-app activities. Both NetScope and this work detect smartphone user activities, but this work still relies on server name resolution and requires network traffic to have similar temporal order to the traffic signatures, which will hardly be the case due to the transient connectivity challenge.

7 Conclusion

Modern, highly specialized mobile apps leave behind fingerprints on their wireless network traffic's behavior. We have presented NetScope, a tool that leverages traffic behavioral clues to detect in-app user activities. NetScope automatically builds models for different activities from their measured traffic behaviors. The models can then be deployed in a NetScope detection module to perform inference of user activities with high accuracy *by observing only IP packet headers*, for both Android and iOS devices.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and suggestions. This work was supported in part by NSF under Award 1409668 and a gift from Cisco Systems. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

References

[1] AMALFITANO, D., FASOLINO, A. R., TRAMONTANA, P., DE CARMINE, S., AND MEMON, A. M. Using gui ripping for automated testing of android applications. In *Proc. IEEE/ACM International Conference on Automated Software Engineering* (2012).

[2] AZIM, T., AND NEAMTIU, I. Targeted and depth-first exploration for systematic testing of android apps. In *Proc. Conference on Object-Oriented Programming Systems, Languages, and Applications* (2013).

[3] BEYMER, M. R., WEISS, R. E., BOLAN, R. K., RUDY, E. T., BOURQUE, L. B., RODRIGUEZ, J. P., AND MORISKY, D. E. Sex on demand: geosocial networking phone apps and risk of sexually transmitted infections among a cross-sectional sample of men who have sex with men in los angeles county. *Sexually Transmitted Infections* (2014).

[4] CAI, X., ZHANG, X. C., JOSHI, B., AND JOHNSON, R. Touching from a distance: Website fingerprinting attacks and defenses. In *Proc. CCS* (2012).

[5] CHEN, S., WANG, R., WANG, X., AND ZHANG, K. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Proc. IEEE S&P* (2010).

[6] COMPARETTI, P. M., WONDRACEK, G., KRUEGEL, C., AND KIRDA, E. Prospex: Protocol specification extraction. In *Proc. IEEE S&P* (2009).

[7] CONTI, M., MANCINI, L. V., SPOLAOR, R., AND VERDE, N. V. Can't you hear me knocking: Identification of user actions on android apps via traffic analysis. In *Proc. ACM Conference on Data and Application Security and Privacy* (2015).

[8] CONTI, M., MANCINI, L. V., SPOLAOR, R., AND VERDE, N. V. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions On Information Forensics and Security 11*, 1 (2016).

[9] COULL, S. E., AND DYER, K. P. Traffic analysis of encrypted messaging services: Apple imessage and beyond. *ACM SIGCOMM Computer Communication Review 44*, 5 (2014).

[10] CUI, W., KANNAN, J., AND WANG, H. J. Discoverer: Automatic protocol reverse engineering from network traces. In *Proc. USENIX Security Symposium* (2007).

[11] DAI, S., TONGAONKAR, A., WANG, X., NUCCI, A., AND SONG, D. Networkprofiler: Towards automatic fingerprinting of android apps. In *Proc. IEEE INFOCOM* (2013).

[12] DUG SONG AND CONTRIBUTORS. dpkt. `https://dpkt.readthedocs.org/en/latest/index.html`, 2015.

[13] DYER, K. P., COULL, S. E., RISTENPART, T., AND SHRIMPTON, T. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *Proc. IEEE S&P* (2012).

[14] FALAKI, H., LYMBEROPOULOS, D., MAHAJAN, R., KANDULA, S., AND ESTRIN, D. A first look at traffic on smartphones. In *Proc. ACM Internet Measurement Conference* (2010).

[15] GONG, X., KIYAVASH, N., AND BORISOV, N. Fingerprinting websites using remote traffic analysis. In *Proc. CCS* (2010).

[16] HAFFNER, P., SEN, S., SPATSCHECK, O., AND WANG, D. Acas: automated construction of application signatures. In *Proc. ACM SIGCOMM Workshop on Mining Network Data* (2005).

[17] HERRMANN, D., WENDOLSKY, R., AND FEDERRATH, H. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proc. ACM Workshop on Cloud Computing Security* (2009).

[18] LI, W., AND MOORE, A. W. A machine learning approach for efficient traffic classification. In *Proc. IEEE Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (2007).

[19] LIBERATORE, M., AND LEVINE, B. N. Inferring the source of encrypted http connections. In *Proc. CCS* (2006).

[20] MA, J., LEVCHENKO, K., KREIBICH, C., SAVAGE, S., AND VOELKER, G. M. Unexpected means of protocol inference. In *Proc. ACM Internet Measurement Conference* (2006).

[21] MILLER, B., HUANG, L., JOSEPH, A. D., AND TYGAR, J. D. I know why you went to the clinic: Risks and realization of https traffic analysis. In *Privacy Enhancing Technologies* (2014).

[22] MOORE, A. W., AND ZUEV, D. Internet traffic classification using bayesian analysis techniques. In *Proc. ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.* (2005).

[23] NIELSEN. An Era of Growth: The Cross-Platform Report Q4 2013. `http://www.nielsen.com/us/en/insights/reports/2014/an-era-of-growth-the-cross-platform-report.html`, 2014.

[24] PANCHENKO, A., NIESSEN, L., ZINNEN, A., AND ENGEL, T. Website fingerprinting in onion routing based anonymization networks. In *Proc. ACM Workshop on Privacy in the Electronic Society* (2011).

[25] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* (2011).

[26] RI.GOV PRESS RELEASES. Rhode Island HEALTH Releases New Data on Infectious Syphilis, Gonorrhea, and HIV. `http://www.ri.gov/press/view/24889`, 2015.

[27] SAPIO, A., LIAO, Y., BALDI, M., RANJAN, G., RISSO, F., TONGAONKAR, A., TORRES, R., AND NUCCI, A. Per-user policy enforcement on mobile apps through network functions virtualization. In *Proc. ACM Workshop on Mobility in the Evolving Internet Architecture* (2014).

[28] SCHNEIDER, F., FELDMANN, A., KRISHNAMURTHY, B., AND WILLINGER, W. Understanding online social network usage from a network perspective. In *Proc. ACM Internet Measurement Conference* (2009).

[29] STÖBER, T., FRANK, M., SCHMITT, J., AND MARTINOVIC, I. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Proc. ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2013).

[30] SUN, Q., SIMON, D. R., WANG, Y.-M., RUSSELL, W., PADMANABHAN, V. N., AND QIU, L. Statistical identification of encrypted web browsing traffic. In *Proc. IEEE S&P* (2002).

[31] TONGAONKAR, A., DAI, S., NUCCI, A., AND SONG, D. Understanding mobile app usage patterns using in-app advertisements. In *Passive and Active Measurement* (2013).

[32] VERDE, N. V., ATENIESE, G., GABRIELLI, E., MANCINI, L. V., AND SPOGNARDI, A. No nat'd user left behind: Fingerprinting users behind nat from netflow records alone. In *Proc. IEEE International Conference on Distributed Computing Systems* (2014).

[33] WANG, T., CAI, X., NITHYANAND, R., JOHNSON, R., AND GOLDBERG, I. Effective attacks and provable defenses for website fingerprinting. In *Proc. USENIX Security* (2014).

[34] WANG, Y., YUN, X., SHAFIQ, M. Z., WANG, L., LIU, A. X., ZHANG, Z., YAO, D., ZHANG, Y., AND GUO, L. A semantics aware approach to automated reverse engineering unknown protocols. In *Proc. IEEE International Conference on Network Protocols* (2012).

[35] WANG, Y., ZHANG, Z., YAO, D. D., QU, B., AND GUO, L. Inferring protocol state machine from network traces: a probabilistic approach. In *Proc. International Conference on Applied Cryptography and Network Security* (2011).

[36] WEI, X., GOMEZ, L., NEAMTIU, I., AND FALOUTSOS, M. ProfileDroid: multi-layer profiling of android applications. In *Proc. Annual International Conference on Mobile Computing and Networking* (2012).

[37] WHITE, A. M., MATTHEWS, A. R., SNOW, K. Z., AND MONROSE, F. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *Proc. IEEE S&P* (2011).

[38] WORKPLACE FAIRNESS. Your Rights: Retaliation - Political Activity. `http://www.workplacefairness.org/retaliation-political-activity`, 2015.

[39] WRIGHT, C., MONROSE, F., AND MASSON, G. M. Hmm profiles for network traffic classification. In *Proc. ACM Workshop on Visualization and Data Mining for Computer Security* (2004).

[40] WRIGHT, C. V., BALLARD, L., COULL, S. E., MONROSE, F., AND MASSON, G. M. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *Proc. IEEE S&P* (2008).

[41] WRIGHT, C. V., BALLARD, L., MONROSE, F., AND MASSON, G. M. Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob? In *Proc. USENIX Security* (2007).

[42] WRIGHT, C. V., MONROSE, F., AND MASSON, G. M. On inferring application protocol behaviors in encrypted network traffic. *The Journal of Machine Learning Research* (2006).

[43] XU, Q., ERMAN, J., GERBER, A., MAO, Z., PANG, J., AND VENKATARAMAN, S. Identifying diverse usage behaviors of smartphone apps. In *Proc. ACM Internet Measurement Conference* (2011).

[44] XU, Q., LIAO, Y., MISKOVIC, S., MAO, Z. M., BALDI, M., NUCCI, A., AND ANDREWS, T. Automatic generation of mobile app signatures from traffic observations. In *Proc. IEEE INFOCOM* (2015).

[45] ZHANG, F., HE, W., LIU, X., AND BRIDGES, P. G. Inferring users' online activities through traffic analysis. In *Proc. ACM Conference on Wireless Network Security* (2011).

[46] ZHOU, X., DEMETRIOU, S., HE, D., NAVEED, M., PAN, X., WANG, X., GUNTER, C. A., AND NAHRSTEDT, K. Identity, location, disease and more: Inferring your secrets from android public resources. In *Proc. CCS* (2013).