AntMonitor: A System for Monitoring from Mobile Devices

Anh Le
CalIT2, UC Irvine
anh.le@uci.edu

Janus Varmarken
IT Univ. of Copenhagen
janv@itu.dk

Simon Langhoff
IT Univ. of Copenhagen
siml@itu.dk

Anastasia Shuba
CalIT2, EECS, CPCC
UC Irvine
ashuba@uci.edu

Minas Gjoka
CalIT2, UC Irvine
mgjoka@uci.edu

Athina Markopoulou
CalIT2, EECS, CPCC
UC Irvine
athina@uci.edu

ABSTRACT

We propose AntMonitor – a system for passive monitoring, collection and analysis of fine-grained, large-scale packet measurements from Android devices. AntMonitor is the first system of its kind that combines the following properties: (i) it provides participating users with fine-grained control of which data to contribute; (ii) it does not require administrative privileges; (iii) it supports client-side analysis of traffic; and (iv) it supports collection of large-scale, fine-grained, and semantic-rich traffic. The first three properties benefit mobile users, by giving them control over their privacy while also enabling a number of services to incentivize their participation. The last property makes AntMonitor a powerful tool for network researchers who want to collect and analyze large-scale, yet fine-grained mobile measurements. As a proof-of-concept, we have developed and deployed a prototype of AntMonitor, and we have used it to monitor 9 users for several months. AntMonitor has high network throughput while incurring lower CPU and battery costs than existing mobile monitoring systems. Our preliminary experience with the prototype demonstrates its capabilities and its potential for enabling several research activities, including network measurement from the edge, classification of mobile traffic, and detection of privacy leakage and other malicious behaviors.

CCS Concepts

•**Networks** → **Network monitoring; Mobile networks;**

Keywords

Network Monitoring; Mobile Networks

This work has been supported by NSF Awards 1228995 and 1028394. Varmarken and Langhoff were visiting UCI when this work was conducted.

C2B(1)D'15, August 17–21, 2015, London, United Kingdom
 ISBN 978-1-4503-3539-3/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2787394.2787396

1. INTRODUCTION

Mobile devices, such as smart phones and tablets, have become ubiquitous. With multiple wireless interfaces, including Wi-Fi and 3G/4G, these devices have persistent Internet connectivity throughout the day. In fact, the amount of traffic generated by these devices has grown rapidly in recent years and is expected to grow by 10 times in the next 5 years [1]. As a result, collecting and studying mobile network traffic has become a critical task in network infrastructure planning and Internet measurement research.

There has been a rich body of literature that studies mobile network traffic [2, 3, 4, 5, 6]. These studies often examine network traffic traces, which typically fall in one of these two categories: either large-scale but coarse-grained traces obtained in the middle of the network, *i.e.*, traces from Internet Service Providers (ISP) [2, 3]; or fine-grained but small-scale traces from a limited set of users [4, 5]. These limitations (*i.e.*, coarse-grained or small-scale) as well as privacy concerns for participating users, have held back research progress in this area.

To this end, we present a system, called AntMonitor[1], for passive monitoring, collection, and analysis of actual network traffic of Android devices. In contrast to existing approaches that use off-the-shelf software [7], we have designed and implemented AntMonitor from scratch. The AntMonitor system includes three components: an Android application, AntClient, and two server applications, AntServer and LogServer. The design aims specifically at enabling collection of large-scale and fine-grained mobile network traffic, in a way that benefits both the participating users and the crowd-sourcing system, as discussed next.

Objective 1: Large-Scale Measurements:

Compatibility and Performance: AntMonitor utilizes the public Virtual Private Network APIs [8] provided by the Android OS (versions 4.0+) and is compatible with more than 94% of Android devices today [9] without requiring administrative privileges (root access). Furthermore, AntMonitor is carefully implemented to achieve high network performance with low overhead. The system is also designed to scale to support tens of thousands of users.

[1]The name AntMonitor is inspired by the *Anteater* (the mascot of UC Irvine, `uci.edu/peter`) and also by crowdsourcing monitoring to a large number of mobile devices that work together like *ants*.

Privacy Control: To facilitate wide user adoption, AntClient is designed to provide users with fine-grained control over which data to contribute. In particular, they can choose specific applications and either full packets or just packet headers to contribute. To the best of our knowledge, AntMonitor is the first system capable of offering this level of privacy control.

Incentives for Participation: In order to attract a large number of users, AntMonitor is designed to be able to offer users with a variety of services including performance (*e.g.*, advertisement blocking) and security and privacy enhancement (*e.g.*, preventing leakage of private information). Furthermore, these could be done completely at the client side, thanks to the custom built AntClient. For instance, a user can protect her email address from leaking without the need to disclose her email address to any remote server.

Objective 2: Fine-Grained Information:

Full Packet Capture: AntMonitor supports the collection of all IP packets on an Android device, including both incoming and outgoing traffic. However, the user may decide to contribute all or part of these packets to the repository, depending on their privacy preferences.

Flexible Annotation: AntMonitor collects packet traces in PCAP Next Generation format [10]. By supporting this format, the system is capable of collecting arbitrary information alongside with the raw packets. This is very important because, in many cases, the side information may only be collected accurately at the client side and could play a critical role in subsequent analyses. For example, AntMonitor currently collects names of applications that are associated with packets, which provide the ground truth for application classification (see Section 5.3).

Objective 3: Making it Attractive for Users: AntMonitor lends itself to crowdsourcing by offering a combination of *incentives* (*e.g.*, the collected data can enable detection of privacy leakage and other malicious behaviors), fine-grained *privacy control* (*e.g.*, the user can select which applications to monitor, and whether to log full packets, headers-only or even meta-data), *ease of use* (*e.g.*, application does not require rooting the phone, simple interface with minimal configuration, service working seamlessly in the background) and *high performance* (*i.e.*, modest CPU and battery penalties that do not affect user experience, see Section 4.2).

As a proof-of-concept, we implemented a prototype of AntMonitor and we have evaluated its performance using synthetic and real-life usage scenarios. Results show that AntMonitor achieves high network throughput performance while incurring less CPU and battery overhead than existing state-of-the-art mobile monitoring systems [7, 11] (see Section 4). We have deployed the system and monitored volunteering students at UCI for several months. We report the results for a pilot measurement period that spans 5 weeks and 9 users. This experiment demonstrates the capabilities of AntMonitor and more importantly, its potential to contribute to the research progress of many applications, including classification of mobile applications and detection of leakage of private information (see Section 5).

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the design and implementation of AntMonitor. Section 4 describes the experiments and reports the evaluation results. Section 5 describes observations from the pilot deployment of AntMonitor and gives preliminary results from applying the AntMonitor system to research directions, such as network monitoring, privacy leakage detection, and traffic classification. Section 6 concludes the paper.

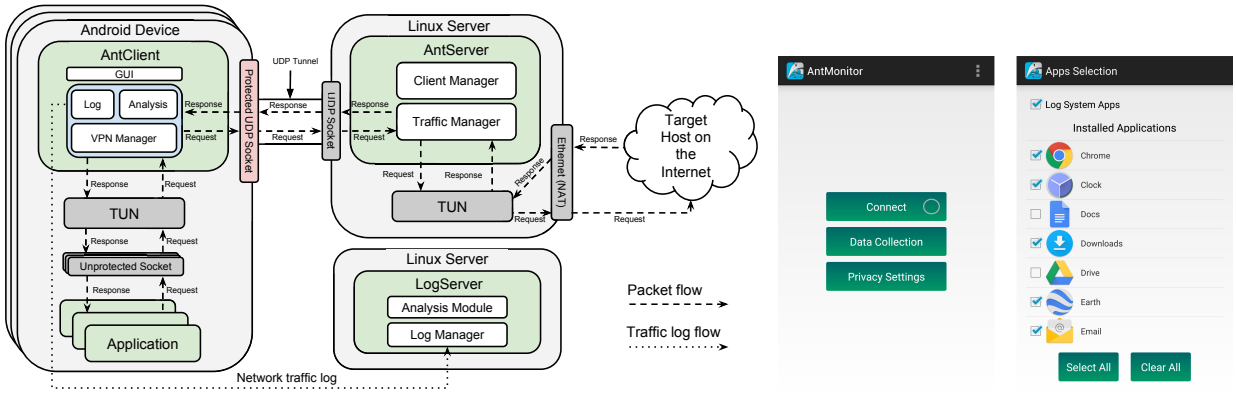
2. RELATED WORK

There is a large amount of work on collecting and analyzing mobile network traffic [2, 3, 4, 5, 6, 12, 13, 14]. Here we only discuss VPN-based approaches [7, 11] as they are directly comparable to AntMonitor. VPN approaches have admittedly several weaknesses: they alter the path of the packets through a VPN server and introduce additional processing to the packets. However, they do allow for passively capturing actual network traffic on the device. Most importantly, they work on most mobile devices today without significantly impacting the user experience. All things considered, we find this approach most suitable to support our identified objectives.

In **server-based VPN** approaches, packets are collected and processed at the VPN server. Disadvantages of this approach include lack of client-side annotated information (*i.e.*, there is no ground truth when mapping from packets to applications at the server), limited privacy protection (*i.e.*, the data is routed and logged at the same server), and complex control mechanisms (*i.e.*, the client has to communicate the selections of functionalities, *e.g.*, ad blocking, to the server). As a representative of this approach, in Section 4, we consider Meddle [7].

In **client-based VPN** approaches, the client establishes a VPN service on the phone to intercept all IP packets. For the captured outgoing packets, it extracts the content and sends them through newly created *protected* UDP/TCP sockets [8] so as to reach Internet hosts. Essentially, the client performs a layer-3 to layer-4 translation per packet for outgoing traffic and vice versa for incoming traffic. This client-based approach does not require a VPN server, but it may have high overhead due to the need to maintain state per connection and additional processing per packet. This affects the network throughput as we show in Section 4, where we consider tPacketCapture [11], an application currently available on Google Play, as a representative of this approach.

We consider AntMonitor to be a **hybrid VPN** approach as it supports both client and server-side analyses, thus combining the best of both worlds. Analyses on the client can (i) protect users in real time, (ii) provide fine privacy control as the user can select which data to contribute, and (iii) provide ground truth mapping of packets to applications and further annotate the traces with rich context information available on the device. Analyses on the server can be applied on a large crowd-sourced dataset, thus having a more complete global view than each individual device, and could also be more complex, unconstrained by mobile CPU or battery.



(a) (b)

Figure 1: (a) AntMonitor System Overview and (b) Screenshots of AntClient.

3. SYSTEM ARCHITECTURE

3.1 Design Overview

The AntMonitor system consists of three components: AntClient, AntServer, and LogServer. Fig. 1(a) shows how the three work together.

Traffic Interception. AntClient establishes a VPN service on the device. This VPN service creates a virtual (layer-3) TUN interface [8] and updates the routing table so that all outgoing traffic, generated by any application on the device, is sent to the TUN interface. AntClient then routes the packets to their target hosts. When a host responds, its packets will be routed back to AntClient, as described below. AntClient then sends the response packets to the applications by writing them to the TUN interface.

Traffic Routing. AntClient sends the packets arrived at the TUN interface through a UDP socket to AntServer, which will further route the packets. To avoid having the outgoing packets of this socket looped back to the TUN interface, AntClient uses a *protected* UDP socket [8]. AntMonitor uses UDP packets for tunneling because it is closest to IP packets (stateless) so as to reduce overhead.

AntServer is configured to also have a TUN interface and have IP Masquerade enabled (packet forwarding with Network Address Translation). When AntServer receives a tunneled packet, it unwraps the packet and writes it to the TUN interface. The packet is then forwarded to the target Internet host of the packet. With IP Masquerade, this packet's source IP, which is a custom IP assigned to AntClient's TUN interface, is translated to the AntServer's IP, and its source port is also translated to a custom AntServer's port. This translation is to ensure that all responses from the Internet hosts are sent to AntServer.

When there is a response packet sent by the target host, it arrives at AntServer, and then the reverse translation is performed: the packet's destination IP and port (AntServer's IP and custom port) are translated back to the original AntClient's TUN interface IP and port. The packet is then available for reading from the TUN interface. Based on AntClient's TUN interface IP, AntServer then sends this response packet through the appropriate UDP tunnel.

Traffic Logging. AntClient saves the intercepted outgoing and incoming packets in log files in PCAPNG format [10]. These log files are uploaded to LogServer for subsequent analyses at a later time: when the device is charging and has Wi-Fi or when explicitly requested by the user.

Privacy Protection. When designing AntMonitor, we explicitly provide routing and logging functionalities using two separate servers. This separation is to provide transparency, fine-grained data collection, and enhanced privacy protection: (i) AntServer only routes traffic and must not log any traffic (similar to most popular VPN services [15]), and (ii) LogServer must only have access to the information explicitly allowed by the user (*i.e.*, user must be able to choose which applications to log).

3.2 Android Application: AntClient

AntClient is an Android application that does not require root access and is compatible with more than 94% of Android devices today (OS versions 4.0+). The application is implemented as a VPN service and works seamlessly in the background of the device, *i.e.*, the user is able to use her favorite applications as usual while running the service. AntClient's main functionality is to log, analyze, and route network traffic. Unlike existing work [7] that uses an off-the-shelf VPN client [16], we implemented AntClient from scratch. This gives us the opportunity to simplify its design by stripping away non-critical yet cumbersome operations, such as, requiring a user to perform authentication, and CPU taxing operations, such as, per packet encryption (which should be optional). In addition, by developing the system from the bottom-up, we can tightly integrate the logging and analysis functionalities into the system. AntClient consists of 4 main components: Graphical User Interface (GUI), VPN Manager, Log Module, and Analysis Module.

Graphical User Interface allows the user to turn the VPN service on and off. The GUI also allows the user to select which applications are permitted to contribute to the data collection. Furthermore, advanced users can also choose to contribute only packet headers or full packets. Fig. 1(b) shows several screenshots of AntClient's GUI.

VPN Manager controls the TUN interface and the UDP tunnel. Its main task is to route packets as described in the previous section. VPN Manager is also responsible for

keeping the VPN service running uninterruptedly despite the volatile mobile environment. In particular, VPN Manager is equipped to deal with various network events, including network errors and switching, *e.g.*, when the device is moving between Wi-Fi and cellular networks.

Log Module writes packets (or packet headers) to log files and uploads the files to LogServer. When packets are intercepted, Log Module first maps them to their corresponding applications. The mapping is done by looking up the UIDs, which represent the applications themselves, of the active network connections available in /proc/net of the device.

Log Module supports storing of additional information of the captured packets by using the PCAP Next Generation format [10]. In our current implementation, we store the raw packets along with their application names. Thanks to PCAPNG flexible format, it is possible to extend the collected information to include rich context, for example, the status of the application associated with the traffic (uptime, foreground vs. background), *etc.* Finally, Log Module periodically uploads the log files to LogServer during idle time, *i.e.*, when the device is charging and has Wi-Fi connectivity.

Analysis Module can accommodate both off-line and online analyses on intercepted packets, such as, detection of leakage of private information, *e.g.*, IMEI, device model, *etc.* The online capability allows it to take action on live traffic, for example, preventing private information from leaking. Since the analyses are done at the client side, private information is never leaked out of the device, setting AntMonitor apart from server-based systems [7].

3.3 Routing Server: AntServer

AntServer manages the clients and routes their traffic. It listens to incoming traffic on a single UDP port. The communication between AntClient and AntServer follows a simple protocol, where messages are prefixed with a single byte header. The value of the header indicates message type.

Client Manager accepts new client connections and keeps track of the connected clients. To initialize the VPN service on the device, AntClient sends a handshake message to AntServer. Upon receiving this message, AntServer responds to AntClient with TUN configuration parameters and starts keeping track of the state of this client. The configuration parameters include a private, unique IP address to assign to the client's TUN interface. To prevent resource leakage, AntServer uses heartbeat messages to keep track of active clients and remove inactive ones.

Traffic Manager routes content messages sent by clients as we discussed in the overview. To support multiple clients, Traffic Manager keeps track of the IP addresses of the clients' TUN interfaces as well as the IP addresses of the clients themselves, *i.e.*, source IP addresses of the wrapped UDP packets sent by the clients. Since the unique IP address of a client's TUN interface will become the target IP address of response packets sent by the Internet hosts to this client, it can be used together with the client's IP to route the response packets back to the correct client.

Scalability. We implement AntServer using Netty, a high performance asynchronous event-driven network application framework [17]. Our implementation is ready for deployment on a private Linux server, Amazon EC2, or Google Compute Engine. Based on the current prototype's CPU, memory, and bandwidth usage, we estimate that a gigabit-network server, *e.g.*, a c4.2xlarge Amazon EC2 machine, could support 500 concurrent users. Thus, supporting a large user base could be done by using a more powerful machine or by deploying more machines.

3.4 Data Collection Server: LogServer

LogServer serves as the central repository to store and analyze all network traffic data. Since LogServer does not have to handle a large amount of live traffic compared to AntServer, it requires less resources to scale. Below are the main components of LogServer.

Log Manager supports uploading files using multipart content-type HTTPS. For each uploaded file, it checks if the file is in proper PCAPNG format. If so, for each client, the manager stores all of its files in a separate folder.

Analysis Module extracts features from the log files and inserts them into a database to support various types of analyses. Compared to the Analysis Module of AntClient, this module has access to the crowdsourced data from a large number of devices, thus suitable for global large-scale analyses. For instance, it could detect global threats and outbreaks of malicious traffic, which may not be possible to perform at the client. In our current implementation, we have analyzed the data to (i) investigate if mobile applications can be classified well based on packet headers and (ii) detect leakage of personal information (see Section 5).

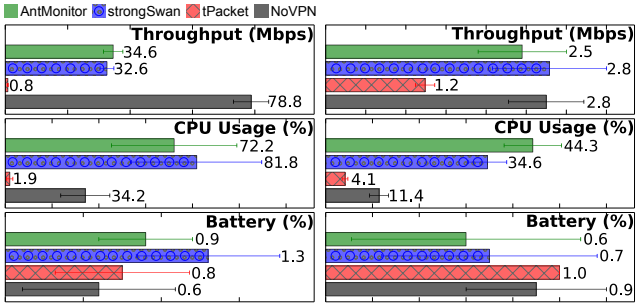
4. PERFORMANCE EVALUATION

All experiments were performed on a Nexus 6 with a Quad-Core 2.7 Ghz CPU, 3 GB RAM, and 3220 mAh battery; the AntServer ran on a Linux machine with 48-Core 800 Mhz CPU, 512 GB RAM, and 1 Gbit Internet; the Wi-Fi network is 5Ghz 802.11ac, and the cellular network is 4G LTE.

4.1 Stress Test

Here we compare AntMonitor against three baselines: NoVpn (not using a VPN service), tPacketCapture VPN v2.0.1 (a representative of the client-based VPN approaches as in Section 2), and strongSwan VPN client v1.4.5, server v5.2.1 (the off-the-shelf software used by Meddle [7], a representative of the server-based VPN approaches as in Section 2).

Setup. For this set of experiments, we used a custom application, called AntEvaluator, that performs downloads and uploads of a large file over HTTP. It can collect various performance statistics, such as, network speed, CPU usage, battery, *etc.* We performed the experiments on both Wi-Fi, with a 1 GB file, and cellular, with a 100 MB file. For a fair comparison, we hosted the VPN servers of strongSwan and AntMonitor on the same machine. The files are hosted on a machine within the same network of the VPN server. During the experiments, AntClient was logging packets of all applications. We ran all experiments with the phone screen turned off and during late night hours in our lab to avoid in-



(a) Wi-Fi (b) Cellular

Figure 2: Performance of AntMonitor, strongSwan, tPacketCapture, and NoVpn when downloading 1 GB file on Wi-Fi 802.11ac and 100 MB file on 4G LTE.

terference. For each experiment, we performed 10 runs and calculated the averages and standard deviations.

Results. Figure 2 shows that on both Wi-Fi and cellular networks, the throughput of AntMonitor is comparable to that of strongSwan and significantly outperforms tPacketCapture. It also shows that using VPN services does affect the throughput. This is because with VPN, packets have additional processing. We observed similar performance for file upload experiments (omitted due to lack of space).

Figure 2(a) also shows that AntClient consumes less CPU and battery than strongSwan on a high-speed Wi-Fi network. The higher CPU usage of strongSwan could be because it has to perform encryption or decryption per packet; however, this overhead is not noticeable on a slower network as shown in Fig. 2(b). Although the CPU usage of 34–82% on Wi-Fi seems high, the maximum CPU usage on the quad-core Nexus 6 is 400%. AntMonitor also achieves similar network latency to that of strongSwan, *e.g.*, on Wi-Fi, AntMonitor and strongSwan both have 12 ms average latency, where the average latency of NoVpn is 8 ms.

4.2 Typical Day Test

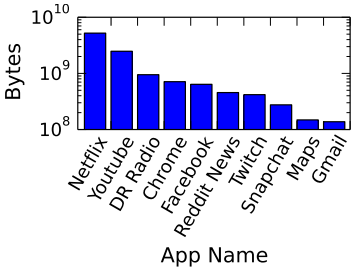
Here we evaluate how AntMonitor impacts the usage experience of a typical mobile user.

Setup. Based on the 2014 Nielsen Survey [18], the 5 most popular mobile application categories are the followings: (i) Search, Portals & Social, (ii) Entertainment, (iii) Communication, (iv) Productivity, and (v) News & Information. Together they take up 92% of the users' application usage time. We extract from the survey the average number of minutes that a user spends on each of these categories per day. We simulate a typical day of a mobile user by using representative applications in these categories. To repeat the experiment multiple times, we record and replay our actions using the Finger Replayer application [19].

More specifically, we simulate 58 minutes of application use in total. This includes 11 minutes of Facebook (Social: profile browsing), 11 minutes of Chrome (Search: web browsing), 21 minutes of YouTube (Entertainment: video streaming), 7 minutes of Gmail (Communication: email viewing, composition, and sending), 5 minutes of Google Keep (Productivity: note reading and writing), and 3 minutes of Reddit News (News: news reading). Each experiment was repeated 3 times for 2 settings: NoVpn and AntMonitor. AntClient logs packets of all applications when used.

	Protocol	Cellular	Wi-Fi
TCP	HTTP	43.0%	53.5%
	HTTPS	51.9%	43.6%
	Other	1.9%	2.6%
UDP	DNS	0.6%	0.1%
	Other	1.4%	0.1%
ICMP		0.2%	0.0%
Total Volume (GB)		1.45	16.07

Table 1: Traffic collected from the user study.



Category	% Bytes
Entertainment	45.5
Media & Video	19.6
Social	7.8
Music & Audio	7.7
Communication	7.6
News & Magazines	4.7
System	2.4
Travel & Local	1.2
Other	3.3

Figure 3: Top 10 popular applications and top categories of traffic collected from the user study.

Results. When on Wi-Fi, the simulation uses on average 13% of the battery with NoVpn and 16% with AntMonitor. AntClient also successfully mapped 99% of network flows to applications. Similarly, when on cellular, the simulation uses on average 14% of the battery with NoVpn and 17% with AntMonitor. (Note that the Nexus 6 has one of the largest batteries, at 3220 mAh.) In summary, these results show that in a typical usage scenario than the stress test, AntMonitor uses only a modest amount of 3% additional battery, which will not significantly affect user experience.

5. USER STUDY

We recruited student volunteers from UC Irvine to use AntClient on their phones. AntMonitor collects the packets of the applications that the volunteers selected and logs them at LogServer. This helped us debug the system over several months and also provided preliminary data that we analyzed. In this section, we present data of 9 volunteers during the period of February 5 – March 15, 2015.

5.1 Data Summary

Table 1 shows the collected data categorized by transport, application protocols, and network connectivity types. In line with previous measurements [7], the majority of the network traffic in smartphones consists of HTTP and HTTPS data. AntMonitor also annotates each packet with the name of its application, and we observed network traffic from 151 applications. Fig. 3 shows the volume of traffic of the top 10 applications and the relative amount of traffic per category.

5.2 Detection of Privacy Leakage

Personally Identifiable Information (PII) is information that can be used to uniquely identify a single person or an individual in a specific context. Such information is sensitive and ideally should not be collected and/or transmitted in plain text by applications. AntClient provides the user with

Personally Identifiable Info.	# Leaking Apps	# Users
IMEI	5	4
Android Device ID	4	6
Phone Number	1	1
Email address	1	1
Location	1	2

Table 2: Number of applications that leaked personally identifiable information in the collected data.

the ability to check whether any of the installed applications are sending her PII out to the Internet. With AntMonitor, checking whether PII is leaked can happen entirely at the client, without the need to either send the PII to any remote server [7], or have a customized OS or rooted device [20].

In our dataset, we tested for leaks of the following PII groups: (i) IMEI, which uniquely identifies a device within a mobile network, and Android Device ID, which is an identification code associated with a device; and (ii) phone number, email address and location, which uniquely associate with users. Table 2 presents the results of our analysis. We observe that 44% and 66% of the users have applications that leak their IMEI and Android Device ID. The majority of leaks are from a small number of applications, and these applications are seemingly harmless, *e.g.*, iWindsurf, a weather application, and Radio FM, an audio streaming application. PII in the second group is rarely leaked, presumably because it is considered sensitive by software developers. However, there are a few exceptions, such as Evite, which leaks email accounts in plain text.

5.3 Application Classification

We use supervised learning methods to build a multi-class model that classifies network flows to specific applications. Building profiles of applications using network-level features can be useful in several ways, *e.g.*, it can assist ISPs who may want to understand and optimize the type of traffic passing through their networks, and it can also facilitate anomaly detection on the device. Our novelty compared to previous approaches [21, 22] is that we use only layer-3 and 4 packet header features, and we have realistic network traffic traces that reflect normal user behavior.

Methodology. We calculated a set of 84 network-level features on both the upstream and downstream flows. These are widely used features from network traffic classification literature and can be partitioned into the following groups: packet length statistics, payload length statistics, inter-arrival time statistics, burstiness, overall flow statistics, and TCP flags. We then performed classification on 70 applications that have at least 30 flows. We used 10-fold cross-validation to avoid overfitting and kept the same proportions of the applications in the testing and training.

Results. We are able to classify a flow to a specific application with F1-score of 70.1% using a Linear SVM. To put this result into context, Meddle [7] reports a 64.1% precision score in classifying flows for the 92 most popular Android applications by using payload features: Host and User-Agent. AppPrint [22] reports that in a large dataset of millions of applications, only 1% of the flows can be classified by using features from HTTP headers. Interestingly, just by using off-the-shelf learning tools, we are already able to classify applications better than state-of-the-art approaches. This is thanks to the fine-grained information available at training: we can associate a packet with the application that generated it.

6. CONCLUSION

In this work, we present AntMonitor – a system for crowdsourcing large-scale, yet fine-grained network measurements from mobile devices. AntMonitor is designed from the bottom up to bring benefits to both the crowdsourcing system, network researchers and the participating users. AntMonitor is designed to scale and is carefully implemented to achieve high network performance with low CPU and battery overhead. Our pilot deployment of AntMonitor shows that it can greatly assist network research activities, including but not limited to, network measurements, detection of malicious behaviors, and traffic classification. Additional materials and a video demo of AntMonitor can be found on our project website [23].

7. REFERENCES

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014-2019. http://goo.gl/Zu8f2r.

[2] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. *IMC'11*.

[3] X. Chen, R. Jin, K. Suh, B. Wang, and W. Wei. Network Performance of Smart Mobile Handhelds in a University Campus WiFi Network. *IMC'12*.

[4] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A First Look at Traffic on Smartphones. *IMC'10*.

[5] N. Vallina-Rodriguez, A. Auçinas, M. Almeida, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. RILAnalyzer: A Comprehensive 3G Monitor on Your Phone. *IMC'13*.

[6] Netalyzr. https://goo.gl/i7HyLU.

[7] A. Rao, A. M. Kakhki, A. Razaghpanah, A. Tang, S. Wang, J. Sherry, P. Gill, A. Krishnamurthy, A. Legout, A. Mislove, and D. Choffnes. Using the Middle to Meddle with Mobile. Technical report, Northeastern University, Dec. 2013.

[8] Android VpnService. http://goo.gl/kV7ZZL, 2014.

[9] Android Versions. developer.android.com/about/dashboards.

[10] PCAPNG File Format. http://goo.gl/y89d9U.

[11] tPacket. www.taosoftware.co.jp/en/android/packetcapture.

[12] PhoneLab, University at Buffalo. https://www.phone-lab.org/.

[13] J. Sommers and P. Barford. Cell vs. WiFi: On the Performance of Metro Area Mobile Connections. *IMC'12*.

[14] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: Multi-layer Profiling of Android Applications. *MobiCom'12*.

[15] Private Internet Access Privacy Policy. http://goo.gl/Yt8jNx.

[16] strongSwan VPN Client. https://goo.gl/okVQYL.

[17] Netty. http://netty.io.

[18] MultiMedia. Smartphones: So Many Apps, So Much Time, 2014.

[19] FRep - Finger Replayer. https://goo.gl/Qtfcva.

[20] W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *OSDI'10*.

[21] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. Networkprofiler: Towards Automatic Fingerprinting of Android Apps. *INFOCOM'13*.

[22] S. Miskovic, G. M. Lee, Y. Liao, and M. Baldi. AppPrint: Automatic Fingerprinting of Mobile Applications in Network Traffic. *PAM '15*.

[23] AntMonitor Project Website. http://antmonitor.calit2.uci.edu.