Android Phone Based Appraisal of App Behavior on Cell Networks

Shaifali Gupta, Rashi Garg, Nikita Jain, Vinayak Naik, and Sanjit Kaul
IIIT-Delhi, New Delhi, India
{shaifali1219, rashi1216, nikita1210, naik, skkaul}@iiitd.ac.in

ABSTRACT

The rapid adoption of smartphones has engendered a large ecosystem of mobile data applications. In fact, a large part of mobile traffic is now data and not voice. Many of these applications, for example VoIP clients, stay active in the background. In the background, they may not communicate large amounts of data. However, their regular bursts of activity can lead to large signaling overheads, wastage of radio resources, and draining of a phone's battery. Signaling overheads can lead to service outage over a large geographical region by overloading the radio network controller (gateway) of the 3G (LTE) network, which is expected to handle signaling from a large number of base stations.

In this work we propose for Android smartphones an on-the-phone mechanism to detect background applications that due to bad design (given the network's settings) or their malicious nature (exploiting the network's settings) lead to above mentioned inefficiencies. Specifically, we propose the metrics of average *energy/byte* and the average time-to-state-promotion (*TSP*) after the Radio Resource Control (RRC) enters the IDLE state. The metrics capture an application's efficiency, which is a function of the network's settings of RRC inactivity timeout values and an application's background activity. The efficacy of these metrics is tested on commonly used Android applications. We also outline a fully functional ready-to-install tool that we developed and used for our studies.

1. INTRODUCTION

Cellular wireless data networks, for example 3G/LTE, have a hierarchical architecture (see Figure 1) in which many base stations may be controlled by a few radio network controllers and/or core network gateways. A RNC in a 3G network (a gateway in LTE) handles signaling and data traffic to and from a large number of mobile phones. While the data path is handled by fast network processors or ASIC(s), the signaling path is handled by the gateway's CPU (slow path) to be able to handle the required computational logic. Increases in signaling volumes can overload this CPU and can cause an outage of data services in a large geographic region [1].

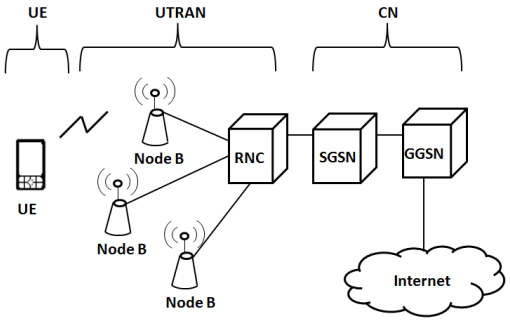

Figure 1: Architecture of a 3G Network. UE is the user equipment. UTRAN is the UMTS radio access network. CN is the core network.

Smartphones have led to an explosion in the amount of mobile data traffic, thereby challenging cellular networks to keep up with user data throughput requirements. Curiously enough, the signaling path is also adversely affected. Among the plethora of applications available to a smartphone user, many of them, for example chat and VoIP, stay active even when in the background, that is when not interacting with the user. Such background applications connect often to the mobile data network [4], albeit only for a short period of time, to send short bursts of data. Such *chatty* behavior leads to frequent allocation and release of radio resources at the mobile network, procedures that are accompanied by a significant amount of signaling between the mobile and the network. Frequent signaling can overload the gateway and in the worst case cause network outage. A recent example of such an outage took place in Japan[1].The mobile network suffered complete network outage for several hours due to frequent control messages from some popular mobile apps.

Recent works, [9], [6], have analyzed such applications. However, they focus on energy efficiency and perceived end user experience, and not on measuring an application's impact on the network's signaling load and energy consumed, for communicating a unit of data. Other commercial efforts that are working toward tackling signaling overload include Seven Networks[2] and The Now Factory[3]. Seven Networks

[1] http://www.techinasia.com/docomo-outage-line/
[2] http://www.seven.com/
[3] http://www.thenowfactory.com/

uses aggregation to alleviate the signaling load on the network. They install a network proxy on the smartphone that buffers network traffic from multiple applications and sends traffic only when a certain minimum amount of payload has accumulated. Unlike them, we provide a mechanism to flag applications that lead to inefficiencies. The Now Factory's approach is not clear.

In our work we say that an application is more *efficient* than another, if the application communicates more data for a given signaling load on the network and energy drain at the phone. We propose the metrics of average energy/byte $\mathbf{E}_B$ and the average time-to-state-promotion (**TSP**) from the RRC IDLE state to discriminate between applications based on how efficient they are. The metrics are motivated by how they capture the behavior of mock (malicious) applications that were designed by us to cause frequent and unnecessary signaling leading to relatively large signaling overheads and energy usage for the amount of user data transmitted, that is poor application *efficiency*. We show that the metrics are able to discriminate between commonly used applications, installed on an Android phone, based on their *efficiency*. All required information is gathered using a tool that runs on an Android smart phone. To the best of our knowledge we are the first to propose such a tool. Our specific contributions are as follows.

1. A tool[4] for Android smartphones is designed to capture state of the RRC protocol and network traffic at the device in real-time. The captures allow us to calculate the proposed metrics.

2. We propose the metrics of average energy/byte and the average time-to-state promotion to quantify an application's *efficiency*.

3. We propose an entirely mobile based framework and show using common applications that it can discriminate between applications and flag those that have a low efficiency (and/or are malicious) and are thus more likely to contribute to unnecessary signaling overloads at the gateway and battery drains at a mobile phone.

The rest of the paper is organized as follows. Section 2 briefly explains relevant parts of the radio resource control protocol. Section 3 describes the tool we have developed. Section 4 explains how we extract values of inactivity timers set by the network. In Section 5 we motivate and explain our metrics and a framework for comparing applications. In Section 6 we detail our methodology. Section 7 discusses results of evaluating popular applications. Section 8 summarizes related works. Finally, in Section 9, we discuss future directions and summarize our contributions.

2. BACKGROUND

This section provides a brief summary of the radio resource controller (RRC) and its states. More details can be found in [2]. The RRC layer at the user equipment (UE) — smartphone in this study — and the radio network controller is responsible for signaling that leads to allocation and release

[4]The tool will be released along with its source code.

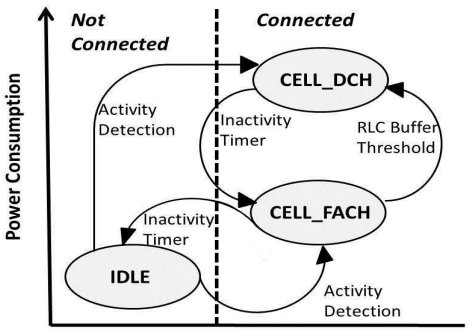

Figure 2: RRC State Machine

of radio resources that allow a mobile phone to communicate over the cellular network.

A UE can either be in idle mode (IDLE) or in connected mode. In the connected mode the UE may be in the CELL_DCH or in the CELL_FACH state. We will drop the prefix "CELL_" for rest of this paper. The states and inter-state transitions are shown in Figure 2.

The UE cannot send or receive data in IDLE. To do so, it needs to transition to one of the connected states. Transition to a connected state, see Figure 2, is initiated when data buffers at the UE or for the UE in the network, exceed certain pre-configured thresholds. Amongst the connected states DCH consumes more energy and provides larger throughput to the UE. It is chosen over FACH for larger buffer occupancies. Transitions to lower energy states are initiated when buffer levels remain low (DCH→FACH) or if no data activity is detected (DCH/FACH→IDLE) for a certain timeout period.

Frequent transitions between the idle and connected states lead to large signaling overheads that can overload the signaling pathways in the network. Also, energy is wasted when the mobile stays in a high energy state waiting for a no-activity timeout to occur.

3. DATA COLLECTION TOOL

Quantifying an application's signaling efficiency requires information about (a) the data packets that were exchanged and (b) the RRC state transitions that took place. Unlike the existing tools, our tool neither requires any special hardware [9] nor retrieves RRC state in an offline manner using a simulator [6]. The functionality of the tool is split into two parts, the *Packet Sniffer* and the *State Logger*. The work flow of the tool is summarized in Figure 3.

3.1 Packet Sniffer: Capturing Data Activity

The packet sniffer uses libpcap [3], which is an open source library for getting user-level packet capture on Unix like systems. We created a library for Android using the libpcap source and the Android NDK. The sniffer is a binary executable for Android and is written in C++. The executable is launched in a shell with superuser privileges.

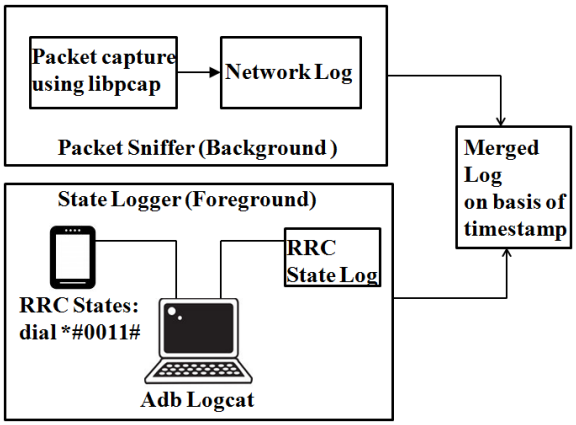

Figure 3: Data Flow Diagram of the Tool. Packet activity is logged using libpcap and works in the background. State changes are logged using secret codes and work in the foreground.

The packet sniffer enables us to capture all uplink and downlink packets associated with a network interface, and stores the packet timestamp and header information corresponding to IP, UDP, TCP, and ICMP protocols, in a text file.

It can capture packets on all the network interfaces, including but not limited to 3G and WiFi. We have tested it successfully for Android version 2.3 and above.

3.2 State Logger: Retrieving RRC State

We adopt a novel approach to find and record the actual RRC state of the device at any given instant. Secret codes are used to open Android hidden menus [8]. The secret codes are manufacturer specific numeric/symbolic sequences, which can be dialed from the device to access hidden menus, run diagnostic tests, and reset parameters of the device. A wide variety of secret codes are available for different manufacturers.

For detecting the state continuously, the phone is connected to a PC via the Android Debug Bridge. The secret code *#0011# (Samsung specific[5]) is dialed to display the RRC states, which are saved at regular intervals using Android `logcat`.

We want to associate the RRC states with network traffic. To do so we begin by launching the packet sniffer in the background and the state logger in the foreground. The packet information obtained from the sniffer and the state information from the state logger is then merged on the basis of timestamps. The merged data provides us with a snapshot of network traffic occurring at the phone and its impact on the device state. This helps us to calculate metrics that quantify application efficiency.

4. DECODING NETWORK PARAMETERS

The inefficiencies that an application can cause are intrinsically related to the inactivity timeouts used by the network, set by the service provider, to go from a high energy to a low energy state. The timeouts used are not publicly available. We infer them using the data collection tool and an Android service that generates UDP packets.

Experiments were conducted to determine DCH→FACH and FACH→IDLE timeout values as described in [5]. The UDP packets generated by the service force the RRC state to transition to DCH.

We note the time the device enters DCH (logged by our tool) and the time of the following transition to FACH. The difference between these times is the DCH→FACH timeout, that is the inactivity time after which the RRC transitions from DCH to FACH. Similarly, the FACH→IDLE timeout is calculated.

We calculated the time out values from two different cellular service providers over multiple state transitions and two different smartphones. We do not find a significant difference in the values used by the two service providers. The values were not phone dependent either. The DCH→FACH timeout was obtained to be about 6 seconds and the FACH→IDLE timeout was obtained to be about 32 seconds. Similar values for DCH→FACH were obtained in [5]. However, they measured a FACH→IDLE timeout of 12 seconds.

5. MEASURING APPLICATION EFFICIENCY

Our approach of discriminating between applications based on their efficiency is best demonstrated by two (mock) malicious applications — SYN Attack and ICMP Attack — that we created to force the RRC state machine between idle and connected states as often as possible.

SYN Attack sends SYN packets every 38 seconds. The periodicity is chosen based on our observations that DCH→FACH timeout is about 6 seconds and FACH→IDLE timeout is about 32 seconds. The choice of 38 seconds implies state demotions from DCH→IDLE are almost immediately followed by data in the buffer leading to a state promotion.

The ICMP Attack is similar in that it sends a ping packet every 38 seconds. The only difference with respect to the SYN Attack is that the SYN Attack is one way communication and generates less traffic. Note that neither of these applications have access to the actual RRC state.

We use our tool to log packet traces for each of these applications and also log the corresponding RRC state transitions. The average energy/byte consumed by the SYN Attack is calculated to be 339.74mJ/byte and the average time to state promotion (TSP) after the RRC goes into IDLE is 2.64s. Details of how these parameters are calculated are given in Section 6. The corresponding values for the ICMP attack are 125.18mJ and 3.79s respectively. The rather small TSP of the applications is not surprising given the periodicity of 38s.

The small TSP implies that signaling to allocate resources that were just released needs to be performed. The smaller the TSP the larger the signaling overheads due to an application. SYN Attack is worse for the network than the ICMP Attack as its energy/byte is much greater. Specifi-

[5]We have tested Samsung Galaxy Y Young, Samsung Galaxy Duos, Samsung Chat, and Samsung Galaxy S Advance.

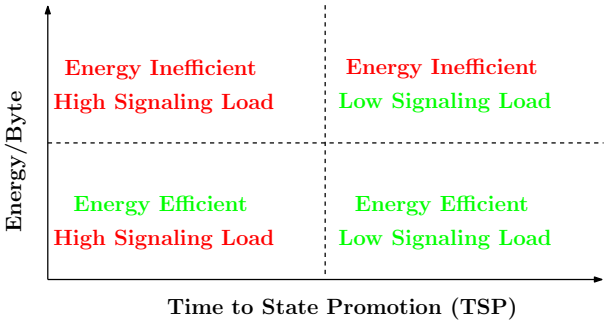

Figure 4: The four quadrants of application behavior. The quadrant an application falls into is a function of its background activity and the network's settings for inactivity timers.

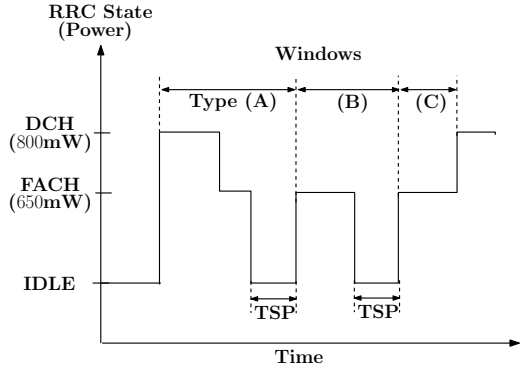

Figure 5: The figure shows a sample trace containing different UE state transitions and their corresponding measurement windows.

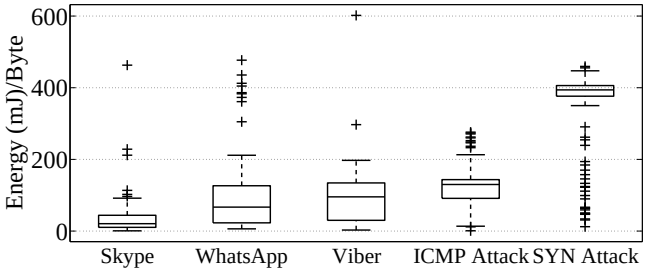

Figure 6: Box plots of Energy/Byte for a few selected applications. Skype does very well on the energy front and is much better than WhatsApp and Viber. Results for all applications we evaluated are not shown here for the sake of clarity.

cally, the amount of data exchanged when the mobile is in connected state is smaller for the SYN Attack. As a result, the energy per transmitted byte expended during connected states for the SYN Attack, which includes the energy that is spent when the network is waiting for inactivity timers to expire, is larger. For a large amount of signaling overheads and energy expenditure the SYN Attack leads to little data being communicated.

Figure 4 shows four quadrants into which applications may be grouped. Applications in the lower right corner are the most efficient given the network's settings of inactivity timers. On the contrary, applications in the upper left corner lead to waste of energy and a lot of signaling overhead for every byte communicated.

6. EXPERIMENT METHODOLOGY

We perform two kinds of experiments. In the first kind, an application's behavior is evaluated in *isolation*. Specifically, we uninstall all other applications and non-essential services from the Android phone. In the second kind, we evaluate application behavior when a mix of applications is installed on the smartphone. For every experiment the packet traces and corresponding state transitions are logged using the tool described in Section 3.

The applications whose behavior we evaluate and compare are Line, WhatsApp, WeChat, Skype, Gmail, and Viber. Line, WhatsApp, and WeChat are instant messaging applications. Skype and Viber are VoIP applications. Gmail is an e-mail application. The similarities in the applications' use cases makes comparing their relative behaviors appropriate. They are also popular amongst Android users.

The two metrics, energy/byte and TSP, are calculated from the log created by the tool. Activities in the log are split into measurement *windows*. A new *window* starts on a state promotion. A state promotion occurs when the RRC state changes from either IDLE→DCH or from IDLE→FACH or from FACH→DCH. The three possibilities are summarized in Figure 5 as window types (A), (B), and (C), respectively. The two metrics are calculated over every window.

In the figure, energy consumed in a window is the area of the curve inside the window. The log also gives us the number of bytes communicated during the window. The two together give us the energy/byte over the selected window. The power consumed in DCH and FACH is set to 800mW and 650mW respectively [6].

The time to state promotion (TSP) after entering the idle state is the time spent in idle just before a window ends (see Figure 5). Note that the window that ends with a FACH→DCH transition does not have a TSP component.

7. RESULTS

We will now show the results of our analysis of the behavior of a set of popular apps that include WhatsApp, WeChat, Viber, Skype, Gmail, and Line. Our experiments confirm that these applications send packets and RRC state transitions take place even when they are in the background and the device screen is off [7].

Figures 6 and 7 show for each application the *box plots*[6] of energy/byte and time to state promotion, respectively. The applications were evaluated in ***isolation***. Each box summarizes data points obtained from about 100 measurement

[6] `http://www.mathworks.in/help/stats/boxplot.html`

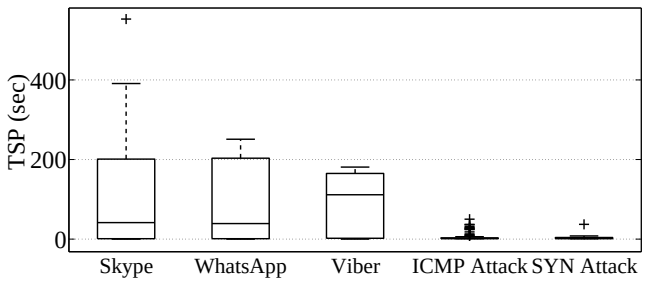

Figure 7: The TSP for selected applications. The spread is similar for Skype, WhatsApp, and Viber. However, Viber has a much larger median TSP. It leads to larger signaling overheads per unit data communicated.

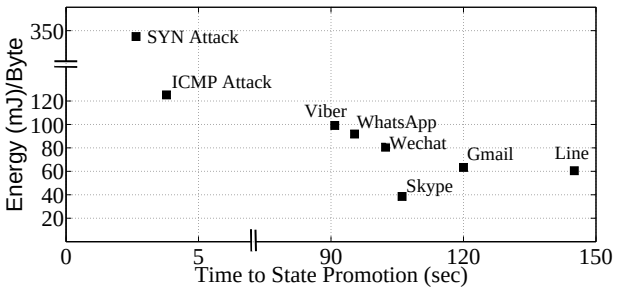

Figure 8: A scatter plot of energy/byte and TSP for all evaluated applications. Viber is the least efficient and its behavior lies in the energy inefficient and high signaling load quadrant. Line is the most efficient and its behavior falls in the energy efficient and low signaling load quadrant. Note the breaks in the axes.

windows.

Note that Skype is about twice as energy efficient (in the background) as compared to the applications Viber and WhatsApp. Not surprisingly, the SYN Attack has the worst energy efficiency.

Now consider the time to state promotion of the applications in Figure 7. The range of TSP values for Skype, Viber, and WhatsApp are about the same. However, the median for Viber is about twice as large as that of Skype and WhatsApp. A larger TSP implies larger signaling overheads on an average per unit data communicated.

While the box plots show the spread of each of the metrics, the scatter plot in Figure 8 helps us place each of the applications into one of the behavior quadrants shown in Figure 4. For each application the scatter plot shows the average energy/byte and average TSP.

All our observations about application behavior are relative to the set we evaluated. Also, remember that the behavior being evaluated is of when the application is running in the background. The application Line is amongst the most energy efficient and also leads to minimal signaling overheads in the network. For our set of applications, Line certainly lies in the lower right quadrant of Figure 4. It has the most desirable behavior. Gmail is not very far from Line and can be placed in the same quadrant. Skype is very energy efficient but leads to larger signaling overheads in comparison to Gmail and Line. It fits well in the lower left quadrant, which groups applications that are energy efficient but lead to signaling overload.

Viber is energy inefficient and also causes signaling overload. WhatsApp is very similar in performance to Viber. For our set, both Viber and WhatsApp lie in the upper left quadrant of energy inefficiency and signaling overload. Finally, WeChat could be placed in either the upper left or the lower left quadrant. While it leads to signaling overload, its energy inefficiency is borderline.

Finally, our mock malicious applications are, as expected, well inside the upper left quadrant. Proximity to such behavior can be an indicator of the application being malicious.

Groups of Applications	Average Energy/Byte (mJ/Byte)	
	Actual	Δ
Gmail+Skype+Line (Baseline)	57.53	—
Gmail+Skype+Line+Viber	82.50	24.97
Gmail+Skype+Line+WeChat	72.57	15.04
Gmail+Skype+Line+WhatsApp	85.68	28.15
Gmail+Skype+Line+Attack App 1	131.10	73.57

Table 1: Average energy/byte for groups of applications

Behavior when groups of applications are installed together: Measuring application behavior in isolation helps us compare applications and flag those with undesirable behaviors. Our preliminary investigation shows that the effect of an application that has undesirable behavior can be detected even when it is installed with others. This is encouraging as it may not always be possible to measure applications in isolation.

Note that the tool has no way of knowing the origin application of a captured packet and therefore cannot know which application caused a state promotion. The group containing the applications Gmail, Skype, and Line are used as a baseline. We quantify the effect of adding an application to this group.

Table 1 shows the average energy/byte measured for different installed groups of applications. The left column under the heading energy/byte tabulates the actual average energy/byte values. The right column tabulates the increase (Δ) in energy/byte when each of Viber, WeChat, WhatsApp, and SYN Attack, are added to the baseline. The energy inefficiencies of Viber, WhatsApp, and SYN Attack emerge clearly. SYN Attack leads to a much larger increase than Viber and WhatsApp, which lead to about the same Δ. This is in line with the observations made for them when they were evaluated in isolation (see Figure 8).

Finally, Table 2 shows the average TSP. The Δ values are inline with expectations set by the scatter plot in Figure 8.

Groups of Applications	Average TSP (seconds)	
	Actual	Δ
Gmail+Skype+Line	87.39	—
Gmail+Skype+Line+Viber	44.82	-42.57
Gmail+Skype+Line+WeChat	45.83	-41.56
Gmail+Skype+Line+WhatsApp	44.30	-43.09
Gmail+Skype+Line+SYN Attack	3.16	-84.32

Table 2: Average TSP for groups of applications

8. RELATED WORK

Previous works to measure RRC parameters observed in UMTS networks have mainly focused on developing mathematical models or adopting indirect techniques to infer the RRC state. The authors in [5] use round trip times of data packets to infer the RRC state. They validate their results by comparing the actual energy consumption of the device in inferred state to the expected consumption in that state. In [6] a tool called ARO (Application Resource Optimizer) is introduced, which uses a simulation-based approach to infer RRC states from packet traces collected on the handset. It is further used to study the inefficiencies of some popular Android apps. Authors in [7] investigate the trade-off between battery efficiency for the end user and signaling load on the side of network operators by studying the behavior of some popular apps. They too attempt to detect the RRC states by using an inference algorithm.

In a much recent work [9] however, the authors presented a novel crosslayer analysis tool - RILAnalyzer, which enables to accurately gather and correlate control plane and user plane events on Android handsets. Currently the tool is supported on rooted Android devices with only Intel/Infineon XGold chipsets. Our tool does not have such dependencies.

While the different techniques listed above have helped the research community to explore the problem of inefficient use of radio resources due to badly designed applications, we also bring to the fore, via mock malicious applications, a possible exploitation of vulnerabilities in RRC state transition machine to cause intentional deterioration in the network performance.

To the best of our knowledge, the work presented in this paper is the first of its kind that can help gauge the inefficiencies (malicious or accidental) of a mobile application by directly looking at the impact of its network activity and the corresponding actual RRC state transitions.

9. DISCUSSION AND CONCLUSIONS

The tool and framework we have proposed can guide application developers toward designing efficient applications. Comparisons with similar applications are also made easy. End users can use the tool to track down applications that take an undesirable toll on the phone's battery and that too when not being actively used. Last but not the least, mobile network operators can use the tool to track behavior of popular applications. Inefficient chatty applications running in the background, not only burden the network, but also are difficult to monetize as the background activity does not involve user interaction.

We plan to analyze the behavior of a very large number of applications. To achieve this, we are working toward automating evaluating the behavior of any application in the Android marketplace. Access to behaviors of a large number of applications will give us the ability to flag applications that are inefficient in comparison to others, say for example, in the same category.

Finally, we are working towards enhancing the tool to capture application activity when in foreground. As a first step the state logger work needs to be modified to work in the background.

In summary, our contributions are as follows. We proposed a general framework that helps classify applications based on their background behavior and its effect on the network. We used the metrics of energy/byte and time to state promotion.

We developed a tool that can log packet activity and RRC state information on an Android phone. The tool is easy to use and is developed using open source libraries.

We showed via example popular applications the ability of the tool and the framework to distinguish between behaviors of applications that have similar use cases. We showed that behavior analysis of an application need not be done in isolation. Finally, we demonstrated the possibility of using the framework to detect malicious applications.

References

[1] A. Gupta, T. Verma, S. Bali, and S. Kaul. Detecting ms initiated signaling ddos attacks in 3g/4g wireless networks. COMSNETS, January 2013.

[2] H. Holma and A.Toskala. *WCDMA for UMTS: Radio Access for Third Generation Mobile Communications.* John Wiley and Sons Inc., New York,USA, 2004.

[3] Libpcap documentation. http://www.tcpdump.org.

[4] M. Paolini and S. Fili. The taming of the app. Sponsored By:Seven Networks(http://www.seven.com/), 2013.

[5] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. ACM SIGCOMM, November 2010.

[6] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. ACM Mobisys, July 2011.

[7] C. Schwartz, T. Hossfeld, F. Lehrieder, and P. Tran-Gia. Angry apps: The impact of network timer selection on power consumption, signalling load, and web qoe. *Journal of Computer Networks and Communications*, 2013(176217), February 2013.

[8] Android secret codes. http://www.digipassion.com/2012/11/samsung-android-mobilephone-secret-codes.html.

[9] N. Vallina-Rodriguez, A. Aucinas, M. Almeida, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. Rilanalyzer: a comprehensive 3g monitor on your phone. ACM Internet Measurement Conference (IMC 2013), October 2013.