Android Phone Based Appraisal of App Behavior on Cell Networks

Shaifali Gupta, Rashi Garg, Nikita Jain, Vinayak Naik, 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. 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. 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. We also outline a fully functional ready-to-install tool that we developed and used for our studies.

Categories and Subject Descriptors

H.4 [**Network Architecture and Design**]: Wireless Communication; D.2.8 [**Testing and Debugging**]: Testing Tools

General Terms

Measurement, Experimentation, Performance

Keywords

Mobile, Cellular Networks, 3G, 4G, LTE, Testing Tools

1. INTRODUCTION

Cellular wireless data networks, for example 3G/LTE, have a hierarchical architecture 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). Increases in signaling volumes can overload this CPU and can cause an outage of data services in a large geographic region [2].

Smartphones have led to an explosion in the amount of mobile data traffic. As a result, the signaling path is also adversely affected. Many user applications, for example chat and VoIP, stay active even when in the background. Such background applications connect often to the mobile data network [6], 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.

The RRC layer [4] at the user equipment (UE) and the radio network controller is responsible for signaling that leads to allocation and release of radio resources. A UE can either be in idle mode (IDLE) or in connected mode. In the connected mode, the UE may be in the DCH (Dedicated Channel) or in the FACH (Forward Access Channel) state. 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 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. 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. Frequent signaling can also cause complete network outage in the worst case. A recent example of such an outage took place in Japan[1].

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. While doing so, we mainly focus on *detection* of undesirable behavior rather than correction. 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. 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.

MOBILESoft'14, June 2–3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2878-4/14/06...$15.00
http://dx.doi.org/10.1145/2593902.2593916

[1] `http://www.techinasia.com/docomo-outage-line/`

2. DATA COLLECTION TOOL AND DECODING NETWORK PARAMETERS

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. The functionality of the tool is split into two parts, the *Packet Sniffer* and the *State Logger*.

2.1 Capturing Data Activity

The *packet sniffer* uses libpcap [5], 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. The packet sniffer enables us to capture all uplink and downlink packets associated with a network interface along with corresponding timestamps.

We used secret codes [1] to find and record the actual RRC state of the device at any given instant. For detecting the RRC state continuously, the secret code *#0011# (Samsung specific[2]) is dialed, and the states are saved at regular intervals using Android `logcat` on the phone. To associate the RRC states with network traffic, 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.

2.2 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[7] used are not publicly available. We infer them using the data collection tool and an Android service that generates UDP packets. 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. Similarly, the FACH→ IDLE timeout is calculated. We calculate the timeout values from two different cellular service providers and two different smartphones. We do not find a significant difference in the values used by the two service providers. The values are not phone dependent either. The DCH→ FACH timeout is obtained to be about 6 seconds and the FACH→ IDLE timeout is obtained to be about 32 seconds. Similar values for DCH→ FACH were obtained in [7]. However, they measured a FACH→ IDLE timeout of 12 seconds.

3. MEASURING APP 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. 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 4. 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.

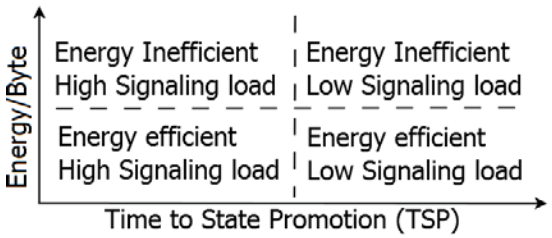

Figure 1: The four quadrants of application behavior. The quadrant an application falls into is a function of its background activity and the network's settings for inactivity timers.

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. Specifically, 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. Figure 1 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.

4. METHODOLOGY FOR EXPERIMENTING

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 2. The applications whose behavior we evaluate and compare are Line, WhatsApp, WeChat, Skype, Gmail, Viber, and Facebook. Line, WhatsApp, and WeChat are instant messaging applications. Skype and Viber are VoIP applications. Gmail is an e-mail application and Facebook is a social networking application. The similarities

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

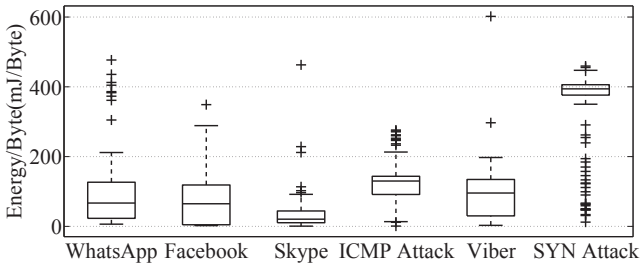

Figure 2: Box plots of Energy/Byte for a few selected applications. Skype does very well on the energy front and is much better than WhatsApp, Viber, and Facebook. Results for all applications we evaluated are not shown here for the sake of clarity.

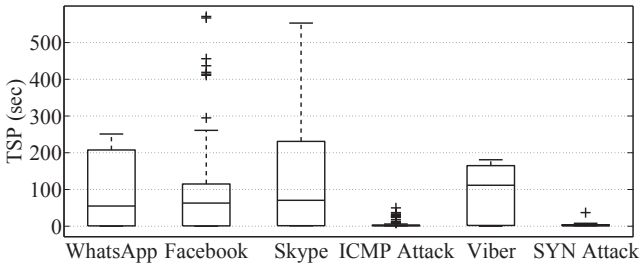

Figure 3: The TSP for selected applications. The spread is similar for Skype, WhatsApp, Viber, and Facebook. However, Viber has a much larger median TSP. It leads to larger signaling overheads per unit data communicated.

in the applications' use cases makes comparing their relative behaviors appropriate.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 two metrics are calculated over every window. The log also gives us the number of bytes communicated during the window and the total time spent in any individual state in 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 [8]. The time to state promotion (TSP) after entering the idle state is the time spent in idle just before a window ends. More details on this can be found in [3].

5. 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, Line, and Facebook. 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 switched off [9]. Figures 2 and 3 show for each application the *box plots*[3] of energy/byte and time to state promotion, respectively. The applications were evaluated in ***isolation***. Each box summarizes data points

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

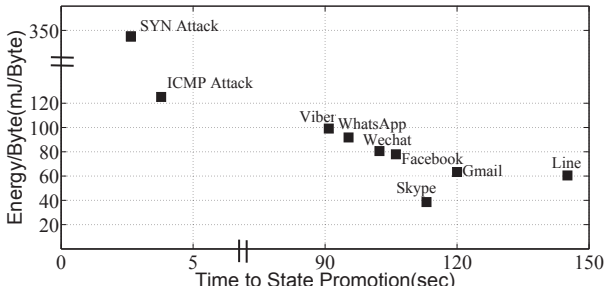

Figure 4: A scatter plot of energy/byte and TSP for all evaluated applications. Viber is the least efficient and its behavior lies in the energy inefficient and high signaling load quadrant. Line is the most efficient and its behavior falls in the energy efficient and low signaling load quadrant. Note the breaks in the axes.

obtained from about 100 measurement *windows*. Note that Skype is about twice as energy efficient (in the background) as compared to the applications Viber, WhatsApp and Facebook. Not surprisingly, the SYN Attack has the worst energy efficiency. Now consider the time to state promotion of the applications in Figure 3. The range of TSP values for Skype, Viber, WhatsApp, and Facebook. However, the median for Viber is about twice as large as that of Skype, WhatsApp and Facebook.

While the box plots show the spread of each of the metrics, the scatter plot in Figure 4 helps us place each of the applications into one of the behavior quadrants shown in Figure 1. 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 1. 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 and Facebook could be placed in either the upper left or the lower left quadrant. While both lead to signaling overload, their 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.

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.

Table 1: Average energy/byte for groups of applications

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 2: Average TSP for groups of applications

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

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. We will add more applications to the set in future.

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 4). Here, it is important to keep in mind that energy/byte values of individual apps are not additive. It is not possible to obtain the energy/byte value of a collection of apps by summing up the values of all the apps in the collection. The important point here is that an app, which shows inefficient behavior individually, brings down the performance of the whole group in which it executes. Finally, Table 2 shows the average TSP. The Δ values are inline with expectations set by the scatter plot in Figure 4.

6. 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 [7] 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 [8] 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 [9] 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 [10] 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.

While the techniques listed above have helped to explore the inefficient use of radio resources caused by 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. For more details on related work, please refer to the technical report[3].

7. CONCLUSIONS AND FUTURE WORK

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. 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.

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.

8. REFERENCES

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

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

[3] S. Gupta, R. Garg, N. Jain, V. Naik, and S. Kaul. Android phone based appraisal of app behavior on cell networks. Technical Report IIITD-TR-2013-003, IIIT-Delhi, October 2013.

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.