RILAnalyzer: a Comprehensive 3G Monitor On Your Phone

Narseo Vallina-Rodriguez*
ICSI
narseo@icsi.berkeley.edu

Andrius Aucinas
Computer Laboratory
University of Cambridge
andrius.aucinas@cl.cam.ac.uk

Mario Almeida
Telefonica Research
4knahs@gmail.com

Yan Grunenberger
Telefonica Research
yan@tid.es

Konstantina Papagiannaki
Telefonica Research
dina@tid.es

Jon Crowcroft
Computer Laboratory
University of Cambridge
jon.crowcroft@cl.cam.ac.uk

ABSTRACT

The popularity of smartphones, cloud computing, and the app store model have led to cellular networks being used in a completely different way than what they were designed for. As a consequence, mobile applications impose new challenges in the design and efficient configuration of constrained networks to maximize application's performance. Such difficulties are largely caused by the lack of cross-layer understanding of interactions between different entities - applications, devices, the network and its management plane. In this paper, we describe RILAnalyzer, an open-source tool that provides mechanisms to perform network analysis from within a mobile device. RILAnalyzer is capable of recording low-level radio information and accurate cellular network control-plane data, as well as user-plane data. We demonstrate how such data can be used to identify previously overlooked issues. Through a small user study across four cellular network providers in two European countries we infer how different network configurations are in reality and explore how such configurations interact with application logic, causing network and energy overheads.

Categories and Subject Descriptors

C.2.3 [**Computer-communication networks**]: Network OperationsNetwork Monitoring

General Terms

Design, Measurement, Performance

Keywords

Energy, Mobile, Radio, Cellular, Networks, RNC

*This work was done while the author was a PhD student at the University of Cambridge.

IMC '13 Barcelona, Spain

http://dx.doi.org/10.1145/2504730.2504764.

1. INTRODUCTION

The success of mobile apps has exposed new issues for end-users (battery life, erratic connectivity), and network providers (coverage and dynamic load management) that were not initially expected on its design. As opposed to wired and WiFi networks, cellular networks have clearly separated control-plane for signaling traffic and user-plane for user traffic respectively. Unfortunately, as previous research has revealed [1,2], the control plane directly impacts on user plane performance and vice-versa.

The way applications use the network affect its control-plane by increasing signaling traffic. The effects also depend on the network configuration. On one hand, the goal is to minimize the network load and improve spectrum efficiency. On the other hand, inappropriate network configurations can decrease the battery life of the handset and increase its communication latency. The two factors are not orthogonal and it is necessary to take into account both the way applications use the network (i.e. user plane) as well as the way network configuration affects applications (i.e. control plane) to alleviate network and energy inefficiencies.

In order to understand application-network dynamics and their inter-dependencies, it is important to follow a cross-layer approach that spans from applications and user events, to the behavior of the control-plane. A major challenge for such analysis is accessing the different layers: neither control-plane information is generally exposed by the radio driver to the mobile OS (only from within the operator), nor user and application events other than network packets are sent to the mobile operator. As a consequence, most of the previous research has been performed either thanks to privileged access to internal and proprietary data from mobile carriers [3–5], expensive test-beds [2] and diagnostic tools [6], or by emulating low-level control-plane events on the terminal [7]. In this paper, we present *RILAnalyzer*, a software and handset-oriented approach that enables gathering of accurate control-plane and user-plane data, including any layer on the protocol stack, *"in the wild"* with real users traffic load, and network configurations.

Firstly, we review the classical vantage points in cellular network research, including low-level diagnostic tools. We discuss their capability and openness in capturing the ground truth in terms of control plane events, user plane, as well as their scope in terms of scale and layers. Previous research is classified based on the type of data and vantage point they use in their analysis. Secondly, we present an

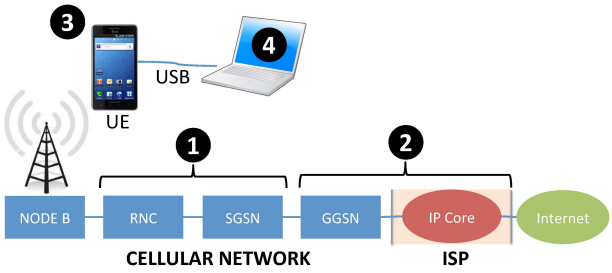


Figure 1: Basic UMTS cellular network deployment and possible probing points.

open source tool motivated by the limitations of these vantage points, we built *RILAnalyzer*, an accurate cross-layer open-source solution capable of reporting and correlating control and user plane events for Android handsets. We demonstrate the capabilities and limitations of our tool by looking at issues that would be difficult to analyze with current methodologies. In particular, we study *i)* the diversity of RNC state machines across different mobile carriers, and *ii)* the network cost associated to TCP operations to reach the back-end infrastructure in popular apps like Facebook, Skype, WhatsApp and Google's Push Notification Service.

2. CELLULAR NETWORK BACKGROUND

As opposed to wired networks, communication on cellular networks induces signaling overhead necessary to control limited spectrum resources and manage devices' power state. Figure 1 shows the basic components of a cellular network. Among all the elements present, the RNC plays a key role managing the spectrum and battery life of the handsets.

The UMTS standard defines three main client power modes known as RRC power states: `IDLE` that corresponds to no connection and low power; `CELL_DCH` (Dedicated Channel) the highest power state with highest throughput and lowest latency; and `CELL_FACH` (Forward Access Channel), a low capacity channel shared across all the mobile users. With the time and successive evolutions, an additional channel known as `CELL_PCH` (Cell Paging Channel) has been introduced between `IDLE` and `CELL_FACH` to enable paging the user equipment at a lower energy cost than `CELL_FACH`. For simplicity, they will be referred to as IDLE, DCH, FACH and PCH respectively. A similar state machine has been defined for LTE standards [8].

The promotions between RRC states are governed by the RNC based on traffic demand, whereas the demotions are defined by inactivity timeouts, unofficially referred to as $T1$ (DCH to FACH), $T2$ (FACH to PCH), and $T3$ (PCH to IDLE) [1]. Their configuration is highly vendor and operator dependent. High inactivity timers can lead to power waste on some devices, as these devices remain at high power states for longer. This has been called as the *tail-energy* [9, 10]. *Fast Dormancy* has been later introduced to minimize the effect of the tail-energy by allowing the handset to demote itself earlier if there is no more traffic [2].

Signaling happens during the promotion and demotion events, and throughout the entire DCH phase as the RRC protocol requires the handset to periodically report its state to the RNC. As signaling traffic is transmitted on a shared channel with limited capacity, its efficient use is critical for correct operation of the network. In fact, if signaling channels get congested, they can even cause network blackouts [2, 11]. Reducing RNC state changes is crucial for cellular operators to reduce spectrum overhead and signaling traffic, but also to extend battery life of mobile handsets. Consequently, understanding the way applications and cellular networks behave is essential to achieve optimal configurations that can satisfy the needs of applications, operating system, cellular networks, and ultimately, the mobile user.

3. ANALYSING CELLULAR NETWORKS

In order to analyze performance of cellular networks and behavior of mobile applications in the network, it is possible to obtain data traces from different vantage points as exemplified in Figure 1 for a 3G network: control-plane traces gathered at network components deployed in commercial networks or dedicated test-beds **(1)**, user-plane information collected from the GGSN or from proxies deployed in the IP core of the ISP **(2)**, user-plane data collected from mobile handsets **(3)**, and chipset-specific engineering/debugging tools running on an external monitoring station to collect *"in situ"* control-plane traces from mobile handsets**(4)**. Their scope and limitations are summarized in Table 1.

Network traces collected from a cellular network have been a convenient way of getting insight into network and application behavior [5]. Although they span across millions of mobile devices, their scope is limited to a single layer of the protocol stack (often HTTP) due to the large amount of signaling and user-plane traffic generated in cellular networks [2]. Gathering extensive control-plane information is only feasible for a limited period of time on a specific element of the cellular network chain like the RNC or a special network testbed **(1)** [5, 12]. Unfortunately, it covers a limited geographic area and logs are only accessible from the mobile operator or network equipment manufacturer.

Most of the previous cellular network studies rely on aggregated application layer data [3, 4, 13], such as web logs, collected and provided by an operator for its Network Operation Center (NOC) **(2)**. As these traces lack control-plane and OS events, researchers have used pre-built models of control-plane operations using advanced ways of modeling RNC state machine based on power measurements and binary search for commercial 3G [1] and LTE networks [8]. RNC models were frequently combined with reverse-DNS techniques to identify which applications generate traffic and to estimate their associated energy consumption and network signaling [3–5, 14]. Such approach can be proven inaccurate on actual mobile apps which combine several online services such as advertising modules, analytics, third-party APIs or even CDNs [3]. Network activity can therefore only be accurately attributed to applications by identifying the process owning a given network socket. We discuss these limitations in more detail in Section 5.

At the terminal level, it is possible to obtain accurate traces of mobile traffic at the user-plane (e.g. using `tcpdump` or `iptables`), user interaction patterns, application and OS events **(3)** [7, 14–16] due to the open-source nature of some OSes (such as Android). Control-plane information can also be extracted from terminals using vendor-specific tools, either licensed such as Qualcomm's XDM [6] **(4)** for Qualcomm, or open-source such as `xgoldmon` [17] for Intel XGold chipsets. Unfortunately, both tools run outside the mobile device using the USB port as a channel for collecting

Vantage point	Control plane	User plane	Scale	Process ID	OS/User events	Access
1. Cellular network components	Ground truth		Large			Operator/Vendor
2. GGSN and IP Core	Inferred	✓	Large	Reverse DNS		Operator/Vendor
3. Mobile OS	Inferred	✓	Medium	✓	✓	Open
4. Engineering tools	Ground truth		Small			Licensed/Open

Table 1: Comparison of the different vantage points used for characterizing cellular networks and mobile application performance along six different axis.

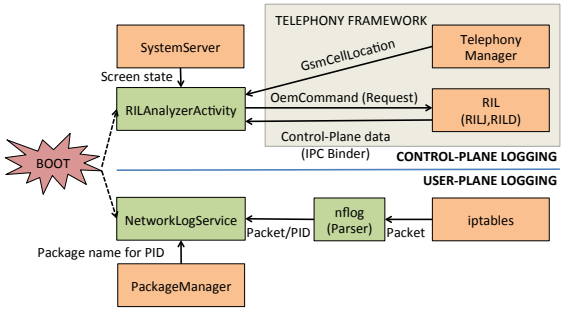


Figure 2: Data flow between the different components

Plane	Data
Control	Cellular technology (GPRS/UMTS/HSPA), RNC state, number of HSPA channels, SNR indicators (e.g. RSCP and EC/IO), Cell ID
Data	Screen state, transport/network layer header, IP header, process ID owning socket

Table 2: Data collected by RILAnalyzer.

data on an external terminal for processing. This makes it difficult to correlate between control-plane information and on-device logs, as well as perform *"in situ"* experiments.

Overall, some vantage points can offer privileged access to the network's ground truth at the expense of gathering limited data across the layers or geographical areas, different operators, number of events captured *etc.* By leveraging terminal capabilities, one could achieve open and decent accuracy across multiple levels (UI events, application, OS events), but even in that case, OS and network design, and current monitoring tools make performing experiments out of the research lab unrealistic. This leaves out an interesting possibility: *how can nearly complete, integrated ground truth knowledge be obtained at the terminal level in the wild?*

4. INTRODUCING RILANALYZER

To overcome the limitations described above, we implemented RilAnalyzer for rooted Android devices with Intel/Infineon XGold chipsets. The tool is publicly available at [18]. Some of the most popular Android devices (e.g. Samsung Galaxy SII/SIII) use this chipset. Figure 2 and Table 2 describe its software architecture and the information it collects. As opposed to other analysis based on data collected from the OS [15, 19], our implementation allows for gathering and correlation of user, application, and OS events with control and user-plane data in a single memory space **from** within the mobile handset.

Control-plane logging: Most mobile platforms are shipped with a dedicated modem chip which runs a real-time firmware in isolation. The main OS of the phone (e.g. Android) communicates with the baseband using the Radio Interface Layer (RIL). A typical Android's RIL spans across three different software sub-components: high-level RILJ (Java module that exposes RIL interface at the Android framework level), the low-level vendor RIL library (that implements vendor-specific messages communication with the modem serial interface), and a RIL Daemon (that runs in memory and translates packets and commands between RILJ and vendor RIL). Although developers can access events such as ongoing calls and the type of cellular network with the public APIs exposed by RILJ, the OS is not capable of accessing directly any control-plane information such as RNC states [1]. This information by default stays in the firmware.

The communication between RILJ and vendor RIL happens through special commands (similar to *AT commands* for PSTN modems) handled by the RILD. Some modems, including those in highly popular Android devices (e.g. Samsung Galaxy SII/SIII) also facilitate a way of sending special, and chipset-dependent commands to the vendor RIL for field-test and debugging measurements reserved to radio engineers. In the case of Samsung's XGold-based handsets, this information is displayed on a foreground application that is launched by entering `*#*#197328640#*#*` on the dialer. Among many other low-level information, it provides access to RF status (e.g. RNC state, HSPA channels), signaling traffic, and control-plane information such as the band of radio access technology (RAT) by converting GUI events into special commands used to poll the radio modem for specific control-plane data.

Although multiple control-plane events can be obtained with such codes, they are not publicly documented. As a result, we had to identify the necessary requests triggering the information we were interested on by adding hooks on RILJ. In particular, we are interested in logging RNC states, which are obtained through the code `*#*#0011#*#*`. Furthermore, any application that interacts with RIL has to use specific radio system permissions, and use non-public RILJ's methods to send `OemCommands` to RIL Daemon on a request/reply basis. To overcome these limitations, we implemented a system tool that runs in the background and polls the modem at the maximum frequency the device responds at, approximately every second (1291 ± 119 ms).

User-plane logging: To log user-plane information we chose to extend and improve `NetworkLog` [20], an open source tool that uses `iptables` with `Nflog` target to log packets in user space. A daemon called `Nflog` reads and parses `netlink` sockets to extract full network and transport layer headers (Figure 2). Once parsed, the information is sent to NetworkLogService (Java background service) to identify the process name for a given PID using the *PackageManager* public API. This solution provides a more accurate traffic to app mapping than other approaches that are based on information available on the `/proc` filesystem [7], especially for short-lived TCP connections and UDP traffic.

4.1 Validation

To verify accuracy of the tool we cross-checked the packets reported by RILAnalyzer against the ones captured by `tcpdump`. We ran 10 series of 100 ICMP packets, at 1 second and 10 ms intervals, DNS lookups to `google.com`, and HTTP requests (BBC's front-page), as well as a full over-night execution with real user-traffic. Exactly the same packets were logged by both RILAnalyzer and `tcpdump`.

Accurate packet timing is a hard problem to solve at the user space: `Nflog` only reports timestamps for outgoing packets as recorded by the Linux kernel. Consequently, they need to be added at the time of processing packets in user-space, which may be significantly later than the actual packet arrival time. This granularity is sufficient for analysing app traffic and control-plane interactions as baseband polling interval is currently limited to 1 second. To record more accurate timing information both modem firmware and Linux kernel require improvements.

Because of the difficulty of obtaining control plane information, we had to verify that RILAnalyzer correctly logs such events in two steps. First, we verified that the vendor-specific tool `xgoldmon` [17] logs control-plane information correctly by comparing its traces against control-plane traces obtained from a cellular network testbed. Given `xgoldmon`'s verbosity when compared against RILAnalyzer (e.g. it logs RRC messages which indicate RNC state changes while RILAnalyzer logs only RNC states), we only had to verify that the polling frequency is compatible with the frequency of occurrence of state changes for basic network operations in several locations under controlled conditions, and that the state reported by the tool is correct.

We used the following technique as both tools cannot run simultaneously on the same handset (due to exclusive use of the same modem interface). We observed that despite the different types of loads (ICMP pings and TCP packets of different size) and different events (3G activation/deactivation) the resulting RNC state events extracted from `xgoldmon` traces were all separated at least by 3-4 seconds, hence the polling frequency of 1 second is not a limiting factor.

4.2 Performance and limitations

Current RILAnalyzer version is limited to Intel Infineon XGold chipsets. Some of the most popular Android devices use the chipsets: Samsung Galaxy SII, SIII, Note 2 and Nexus. We tested our system with the SII and SIII versions. Although the same commands work for LTE networks, we could not test them given limited deployment in Europe. Focusing on single chipsets may induce particular behavior and interaction problems on its own due to Vendor RIL implementation differences, however we perform our case-study (Section 5) on devices shipped with the same radio baseband, hence we still discover network and application differences accurately. As licensed monitoring tools for Qualcomm chipsets are available, we expect finding similar features in other product vendors. Providing support for other chipsets is limited by the effort it takes to reverse-engineer the hidden commands. A more efficient solution would be for vendors to expose the information to the OS.

We observed CPU and memory consumption of RILAnalyzer both idle and under stress conditions on a Samsung Galaxy S2 (Dual-core 1.2 GHz Cortex-A9). The idle experiment was conducted over three hours with Google Services, Skype and Facebook apps active in the background but without any user interaction. Stress conditions were simulated using Speedtest application [21]. During the idle experiment, RILAnalyzer consumed an average of 0.16% of CPU and 22 MB of physical memory. In this period, applications generated periodic traffic that produced variations on the CPU consumption that remained below 10%. In the stress test, at the maximum observed download throughput (+5 Mbps) the maximum observed CPU consumption was 47% using close to 42 MB of physical memory. The high memory load is due to the number of different components logging all aspects of the running systems as well as polling the radio periodically for its state.

The increase is due to the way network packets are logged by our `iptables`-based approach: each packet is duplicated and forwarded to `Nflog`, annotated with the additional information as described above and recorded. Because of platform limitations, the current version of the tool is forced to rely on polling mechanisms to achieve association between applications, network traffic, and cellular state. This adds computational and energy overheads on embedded systems like Android, which have aggressive sleeping policies [22]. Although such overheads are not desirable, they would only be decreased if vendors open their APIs to efficiently gather control plane data within the system.

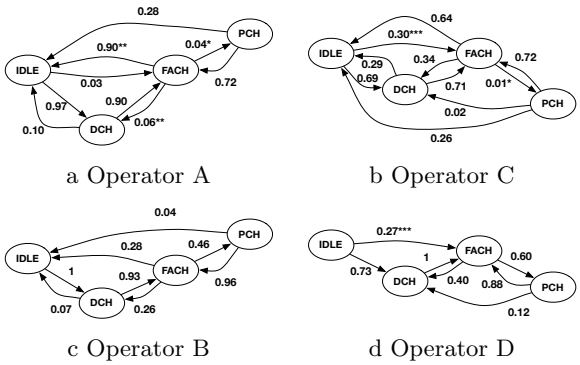
The current version relies on sufficient internal storage to log data for mid-term studies. The volume of logs generated is 62±2 bytes per RNC promotion, and 130±10 bytes per logged packet (including process name). In its current iteration, RILAnalyzer is meant to be used for limited duration studies as a measurement tool in a real environment, rather than a data collection app adopted by a large userbase. Nevertheless, we plan to extend the control-plane traces using other hidden codes, and build an online facility for collecting and processing anonymized data for users, researchers and developers interested in identifying network and apps inefficiencies.

5. CASE STUDIES

In order to demonstrate RILAnalyzer's capabilities, we instrumented eight Samsung Galaxy SII handsets owned by experienced Android users subscribed to different mobile operators. They were asked to run the tool for one complete week, with their normal set of apps. The logged data accounts for more than 1200 hours of mobile activity. During this period of time, 70 applications sent or received more than 2.6M packets, causing +138K RNC transitions (29K promotions to connected states such as FACH (shared) - and DCH (dedicated)). Using the data collected with RILAnalyzer, we characterize the diversity of RNC state machines across the different networks and their effects on resource consumption (Section 5.1). Finally, we evaluate the network costs associated to TCP operations to reach the back-end infrastructure in four popular applications (Section 5.2).

5.1 RNC state machine dynamics

In the past, RNC state machine has been modeled using probes and external power meters [1,9]. Although this identified one of the main energy sinks of mobile systems, they are static and inaccurate as they are obtained *"in situ"*. In contrast, we have used *RILAnalyzer* to record actual device RNC states and demonstrate that RNC state machines are much more diverse, necessitating more complex solutions for improving device performance and energy consumption.



a Operator A b Operator C

c Operator B d Operator D

Figure 3: Observed RNC state transition likelihood

Figure 3 shows such differences. Although RNC promotions are triggered by user traffic, sequences of state transitions are very different.

Figure 3 shows observed transitions between RNC states for 4 different networks, *i.e.* if we look into RNC states in isolation, what is the likelihood of observing a particular transition from one state to another. A number of interesting observations can be made from this figure alone, not least that RNC state machines of these networks are quite different. *Firstly*, it is easy to see that transitions to PCH state in operators A and C are not common (labelled with (*) on figure). PCH is a RNC state introduced in HSPA standards to save battery power during periods of inactivity in the downlink while enabling an efficient paging in case of incoming data. This is likely caused by partial feature deployment in the network thus devices subscribed to these operators without complete PCH state support will experience different behaviors in terms of performance as well as signaling and energy costs.

Secondly, there are only 6% transitions from FACH to DCH vs 90% to IDLE in network A (** on figure). This is caused either by a very short FACH timeout, or by the users' set of applications not generating traffic within this timeout period. The former case would have adverse effects to both the network (largely increased signaling) and the user (control-plane latency due to connectivity reacquisition).

Finally, networks C and D contrast with network A and B in that they have a large proportion of transitions from IDLE to FACH compared to one from IDLE to DCH (*** on figure). This implies that these networks allow devices to use FACH for small messages as opposed to going to DCH for all communication.

We can explain some of these transitions better by also looking at RNC state demotion timeouts as described by Qian *et al.* [1]. The violin plots[1] in Figure 4 show the distribution of the RNC inactivity timeouts T1 (from DCH to FACH) and T2 (from FACH to IDLE or PCH) for the different networks as observed by participating users. We measured the timeouts as time difference between the last packet seen on the interface and the time RNC state is recorded as changed. As shown in Figure 3, a lower density of points indicates less frequent transitions.

Results show a large variability of RNC timeouts. T1 is mostly homogeneous around 5-6 seconds, often reported

[1]Similar to box plots, except that they show the probability density of the data at different values along y axis.

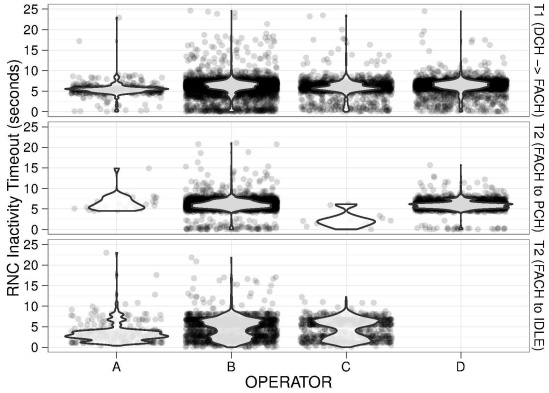


Figure 4: RNC inactivity timers (T1 and T2) in UMTS/HSPA for the different operators

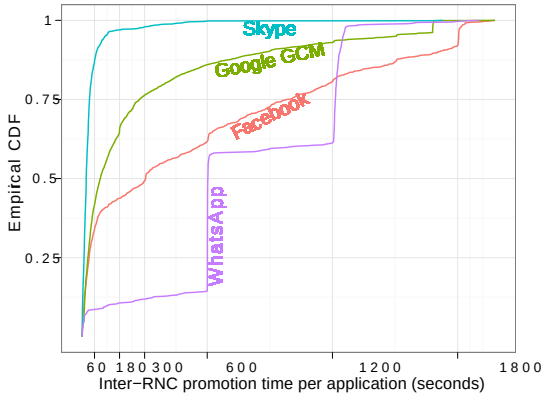


Figure 5: Empirical CDF of inter-RNC promotion time

in previous research work, however there are smaller values (1-2 seconds) that may be associated to Fast Dormancy support at some locations. In fact, inactivity timeouts are highly vendor and operator dependent and the duration can vary depending on traffic demand. It turns out, T1 can be set to different values depending on allocated throughput (8, 16, 32, 64, 128, 256, 384 kbit/s) and is recommended to be set to smaller values for higher throughput to save DCH capacity [23]. Both networks B and C, have two distinctly different values of T2: 6 and 2 seconds, showing fast dormancy behavior. T2 for network A, however, is the lowest one and as discussed above, it causes demotion to IDLE too soon for any promotions back up to DCH to happen. As a result, a lot of signaling and control-plane latency is added.

Both Figure 3 and 4 confirm the low number of transitions from FACH to PCH in networks A and C. This suggests that these transitions occur on rare locations for the users under study. Furthermore, there are no transitions from FACH to IDLE in network D, implying a good use of PCH, which reduces control-plane latency for devices in the network.

5.2 Unveiling cross-layer inefficiencies

To demonstrate the strengths of our approach, we chose to display the costs of basic TCP operations, such as heartbeats and `FIN`s, in terms of RNC state changes on a per-application basis.

Figure 5 shows the empirical CDF of the time between promotion triggered by Facebook, WhatsApp, Skype and Google push notification service (a service shared across

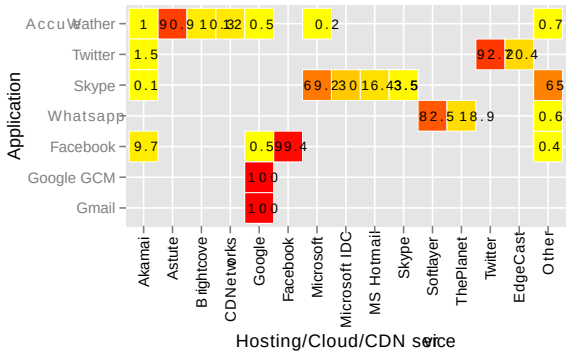


Figure 6: Percentage of observed RNC promotions per application to different backends

multiple applications such as Facebook, and WhatsApp). Here we aggregate inter-promotion times across all users and all networks in the study, therefore ignoring user-specific interaction patterns, but taking into account RNC behavior of each device as recorded by RILAnalyzer. We count promotions to both FACH and DCH states. As we can see, applications have different frequencies in terms of RNC promotion frequency and the steps on the lines (especially clear in the cases of WhatsApp and Facebook) represent periodic messages, such as TCP or HTTP/1.1 keepalives.

Apps have to rely heavily on backend infrastructure, such as CDNs, advertising networks [3], push mechanisms [24], and authentication APIs (e.g. OAuth), often managed by different organizations. An often overlooked fact is that a lot of overheads are exactly due to distributed backends' lack of coordination even in a single app, as indicated by Figure 6. Every long lived TCP connection is kept alive individually. Using reverse DNS to map network traffic to applications would incorrectly attribute it to the apps developed by the same company that owns the backend infrastructure.

Something that is only accurately feasible from within a device (e.g. using RILAnalyzer) is demonstrated in Figure 6: the proportion of RNC promotions in which there is traffic to a particular backend (a few more apps are included for comparison). Each network packet is assigned promotion ID during which it is transmitted and the percentage is obtained by combining organization name obtained from reverse DNS with the promotion ID. Naturally, there are promotions during which traffic goes to multiple backends. This happens when the sum of percentages for an app is more than 100%.

Skype here demonstrates a pathological case due to its P2P nature: RNC promotions are triggered every 30 seconds on average to maintain connections with other connected users. In addition, it relies on distributed services to monitor QoS, billing, and to guarantee reachability and resilience (managed by Microsoft or Skype). As much as 74% of connections are directly to other peers although surprisingly they only account for 57% of packets. Google's GCM, on the other hand, aggregates notifications for multiple applications (Google's own and ones that use it for push notifications) reducing the amortized cost for each application - only one open TCP connection is needed to a single entity serving multiple applications. We can not isolate per-application information from Google GCM protocol and have to analyze the entity as a whole.

Another problem with distributed backends is due to the fact that cellular networks are controlled by middleboxes for security and performance [25]. Such middleboxes impose restrictions on the way applications maintain their connections at the transport [26] and application layer [27]. We found using RILAnalyzer that TCP heartbeats and FINs (messages that are driven by these restrictions) alone account for a significant proportion of RNC promotions (Table 3). In the case of WhatsApp, a large fraction of RNC promotions associated with these operations are due to TCP hearbeats maintaining connections, while the others often also include application-level information.

App	Hearbeats & FIN (%)	Server-side (%)
Facebook	3.2	12.6
Google GCM	6.1	69.6
Skype	15.9	40.2
WhatsApp	47.1	3.0
All apps	41.0	58.9

Table 3: Per-App promotions triggered by TCP operations and initiated by a server

For network-intensive applications such as Skype and push notification mechanisms, a large portion of total promotions are triggered by packets sent from the server-side, indicating where connection maintenance logic is located. On the other hand, applications such as Facebook rely on Google's push notifications, thus reducing the need to maintain their own TCP connections. These observations suggest that to reduce the energy and network overheads of mobile traffic, it is essential to control downlink traffic (e.g. using middleboxes or enhancement proxies) in addition to the classic approach of controlling uplink on the mobile handset [1, 3, 4, 9, 28].

6. CONCLUSION

In this work we presented a tool which facilitates researchers to analyze cellular network issues in a new light without requiring access to the internal components of the cellular network. We prototyped the tool, RILAnalyzer, on the Android platform for popular smartphone devices. RILAnalyzer is meant to be used outside of laboratory environment and therefore allows open accurate mobile device measurement studies across many users, applications, networks and geographical regions. We evaluated performance of the tool and discussed its limitations - some of them inherent to running within a mobile device and not relying on external resources.

We have demonstrated the ability of RILAnalyzer to accurately perform cross-layer analysis on mobile systems. First, we demonstrated important RNC state machines differences across 4 mobile operators in two European countries. Second, to demonstrate the strengths of RILAnalyzer, we exposed inefficient connection maintenance logic for a few popular applications, caused by the large number of backend systems they rely on and sub-optimal use of TCP. We are releasing the tool publicly [18] for the mobile research community to use, extend and improve.

Acknowledgments

This material is based upon work supported by the EPRSC INTERNET Project, the Department Homeland Security under Contract N66001-12-C-0128, and by the NSF under grant 1213157. The authors would also like to thank the volunteers, the anonymous reviewers and our shepherd Aditya Akella (University of Wisconsin-Madison) for constructive feedback on preparation of the final version of this paper.

7. REFERENCES

[1] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3G networks. In *Proceedings of ACM IMC*, 2010.

[2] Nokia Siemens Networks Smart Labs. Understanding smartphone behavior in the network. `http://www.nokiasiemensnetworks.com/sites/default/files/document/Smart_Lab_WhitePaper_27012011_low-res.pdf`, 2011.

[3] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: characterizing mobile advertising. In *Proceedings of ACM IMC*, 2012.

[4] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck. Screen-off traffic characterization and optimization in 3G/4G networks. In *Proceedings of ACM IMC*, 2012.

[5] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Periodic transfers in mobile applications: network-wide origin, impact, and optimization. In *Proceedings of WWW Conference*, 2012.

[6] Qualcomm Extensible Diagnostic Monitor. `http://www.qualcomm.com/media/documents/qxdm-professional-qualcomm-extensible-diagnostic-monitor`.

[7] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of ACM MobiSys*, 2011.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. In *Proceedings of ACM MobiSys*, 2012.

[9] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of ACM IMC*, 2009.

[10] N. Vallina-Rodriguez and J. Crowcroft. Energy management techniques in modern mobile handsets. *Communications Surveys Tutorials, IEEE*, 2013.

[11] Z. Shafiq, L. Ji, A. Liu, J. Pang, S. Venkataraman, and J. Wang. A first look at cellular network performance during crowded events. In *Proceedings of ACM SIGMETRICS*, 2013.

[12] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the top video: the gorilla in cellular networks. In *Proceedings of ACM IMC*, 2011.

[13] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang. A first look at cellular machine-to-machine traffic: large scale measurement and characterization. In *Proceedings of ACM SIGMETRICS*, 2012.

[14] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin. Diversity in smartphone usage. In *Proceedings of ACM MobiSys*, 2010.

[15] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of IEEE/ACM CODESS*, 2010.

[16] P. Abhinav, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof. In *Proceedings of ACM EuroSys*, 2012.

[17] Github. XGoldMon project. `https://github.com/2b-as/xgoldmon`.

[18] RILAnalyzer. `http://rilanalyzer.smart-e.org/`.

[19] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: multi-layer profiling of android applications. In *Proceedings of ACM Mobicom*, 2012.

[20] Github. Network Log. `https://github.com/pragma-/networklog`.

[21] Speedtest android application. `https://play.google.com/store/apps/details?id=org.zwanoo.android.speedtest&hl=en`.

[22] A. Jindal, A. Pathak, Y. C. Hu, and S. Midkiff. Hypnos: understanding and treating sleep conflicts in smartphones. In *Proceedings of ACM EuroSys*, 2013.

[23] Nokia. 3G radio optimisation parameter testing guide. `http://www.scribd.com/doc/103289214/Parameter-Testing-Reference-Quide`.

[24] Google Cloud Messaging. `http://developer.android.com/google/gcm/index.html`.

[25] Z. Wang, Z. Qian, Q. Xu, Z. M. Mao, and Ming Zhang. An untold story of middleboxes in cellular networks. In *Proceedings of the ACM SIGCOMM Conference*, 2011.

[26] F. Busatto. TCP Keepalive HOWTO. `http://tldp.org/HOWTO/TCP-Keepalive-HOWTO`.

[27] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, 1999.

[28] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. TOP: Tail Optimization Protocol For Cellular Radio Resource Allocation. In *Proceedings of IEEE ICNP*, 2010.