MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones

Yuanjie Li[1] Chunyi Peng[2] Zengwen Yuan[1] Jiayao Li[1] Haotian Deng[2] Tao Wang[3]
[1]University of California, Los Angeles [2]The Ohio State University [3]Peking University

ABSTRACT

We design and implement MOBILEINSIGHT, a software tool that collects, analyzes and exploits runtime network information from operational cellular networks. MOBILEINSIGHT runs on commercial off-the-shelf phones without extra hardware or additional support from operators. It exposes protocol messages on both control plane and (below IP) data plane from the 3G/4G chipset. It provides in-device protocol analysis and operation logic inference. It further offers a simple API, through which developers and researchers obtain access to low-level network information for their mobile applications. We have built three showcases to illustrate how MOBILEINSIGHT is applied to cellular network research.

1 Introduction

The cellular network is a "closed" yet critical infrastructure. On one hand, mobile users are increasingly accessing online services through their 3G/4G networks on their smart devices (*e.g.*, smartphones and tablets). The resulting data volume has contributed to 88% of global mobile traffic now, and is projected to reach 97% by 2019, with a tenfold traffic growth [1]. On the other hand, users and devices have very *limited* access to their runtime operations on all cellular protocols. Mobile applications transfer data through the cellular interface via the socket API. Beyond that, the network itself largely remains a blackbox to users.

The lack of open access into fine-grained runtime network operations creates barriers for researchers and developers to accurately understand and refine how cellular protocols operate at the device and inside the network. For example, the device experiences a handoff on the go but has no clue on why it is triggered and whether it is a good decision. In reality, the device has been observed to hand over and get stuck in 2G even when 4G is available. Another real-life instance is that it is not uncommon to take long time to upload a photo or experience a failed call via 4G. It is not clear whether it is caused by poor radio quality or network protocol issues. The list goes on and long.

For a rather closed, large-scale cellular network system, we need both *data* and *analytics* to identify problems and renovate the design. This calls for a community tool suite that can be built and shared together. Such tool could stimulate research on the 3G/4G mobile networks. Ideally, the tool should possess three features simultaneously: (1) It can collect runtime operation traces using commercial off-the-shelf (COTS) devices without extra hardware support; (2) Given the data traces, it provides analytics to extract dynamic protocol behaviors for both common usage settings and abnormal failure cases; (3) The tool offers simple APIs to build applications and the framework can be readily extended. Unfortunately, no such community software tools are available to date. The existing ones cannot meet all three requirements [2–7] (see Table 14 for a comparison summary). Furthermore, operators are reluctant to release the traces collected from the infrastructure side. They also have limited access to the device-side operations.

In this paper, we take the first step to develop MOBILEINSIGHT[1], a software toolwhich enables runtime cellular network monitoring and analytics on COTS smartphones. It aims to satisfy three features above; We seek to overcome the barrier by providing *open access* (in *software*) to *fine-grained* cellular information on 3G/4G protocols; We empower in-device analytics, which not only disclose *what* happens but also shed light on *why* and *how*. The tool is intended as an open platform that is extensible. The goal is to facilitate researchers and developers to *readily* and *quickly* obtain the low-level network information through easy-to-use APIs.

In a nutshell, MOBILEINSIGHT runs as a user-space service on COTS smartphones (root access required for some phone models). It does not require any extra support from operators, or additional hardware (USRP, PC or testing equipments). MOBILEINSIGHT leverages a side channel inside the COTS smartphones, and extracts cellular operations from signaling messages between the device and the network. These control-plane messages regulate essential utility functions of radio access, mobility management, security, data/voice service quality, to name a few. Given these messages, it further enables in-device analytics for cellular protocols. We not only infer runtime protocol state machines and dynamics on the device side, but also infer protocol operation logic (*e.g.*, handoff policy from the carrier) from the network. MOBILEINSIGHT offers a simple API for use and extension. With its simple API, we further describe three example cases to show how MOBILEINSIGHT can be used. These showcases demonstrate how MOBILEINSIGHT benefits end devices in a variety of scenarios, including failure diagnosis, performance improvement, and security loophole detection.

We have implemented MOBILEINSIGHT on Android phones with Qualcomm chipsets (feasibility on iPhones and non-Qualcomm chipsets has been also validated). It currently supports all 3G/4G control-plane protocols for radio resource control, mobility management and session management, as well as certain

[1]http://metro.cs.ucla.edu/mobile_insight/

MobiCom'16, October 03-07, 2016, New York City, NY, USA
 ISBN 978-1-4503-4226-1/16/10 $15.00
DOI: http://dx.doi.org/10.1145/2973750.2973751.

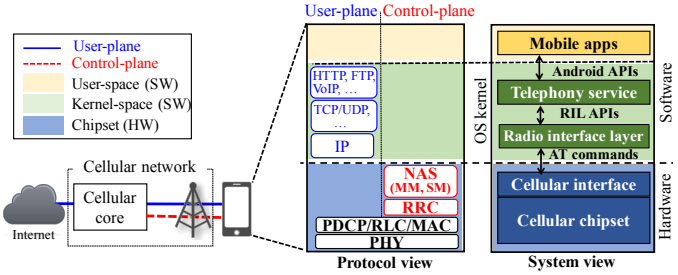


Figure 1: Network architecture (left), protocol stack (middle), and information access at device (right).

Name	Description	Category	System	Standard
CM	Connectivity Management	Data/voice	3G CS	TS24.008 [9]
SM	Session Management	control in	3G PS	TS24.008 [9]
ESM	4G EPS Session Management	NAS	4G	TS24.301 [10]
MM	Mobility Management	Mobility	3G CS	TS24.008 [9]
GMM	GPRS Mobility Management	mgmt. in	3G PS	TS24.008 [9]
EMM	EPS Mobility Management	NAS	4G	TS24.301 [10]
3G-RRC	Radio Resource Control	RRC in	3G	TS25.331 [11]
4G-RRC	Radio Resource Control	AS	4G	TS36.331 [12]
PHY	Physical Layer	Layer 1	4G	TS36.211 [13] TS36.212 [14] TS36.213 [15]
MAC	Medium Access Control	Layer 2	4G[a]	TS36.321 [16]
RLC	Radio Link Control		4G[a]	TS36.322 [17]
PDCP	Packet Data Convergence Protocol		4G[a]	TS36.323 [18]

Table 1: Cellular network protocols and used acronyms.

4G below-IP protocols that convey control information. We have tested MOBILEINSIGHT with 8 carriers: four US carriers (Verizon, AT&T, T-Mobile, Sprint) plus Project Fi [8], and three Chinese operators, using 30 phones from 11 phone models. Our evaluation shows that, MOBILEINSIGHT works well on both high-end and low-end phones using different cellular chipsets and OSes. It logs cellular events and executes analysis with acceptable overhead. It can accurately infer protocol state-machines and operation logics and process 99% of cellular signaling messages in less than 0.8 ms in most cases. The tool itself consumes 1–7% CPU usage, 30MB memory and 3–4% extra energy at maximum. The code and application, as well as the traces we collect through MOBILEINSIGHT, are available to the research community[1].

This work has three main contributions.

1. We present MOBILEINSIGHT, an in-device software tool to monitor, analyze and exploit runtime cellular protocol information on COTS smartphones;
2. We devise side-channel techniques to collect signaling messages for 3G/4G protocols, and design inference techniques to analyze protocol state dynamics and operation logic;
3. We conduct extensive tests to assess its effectiveness, and build three showcase examples to demonstrate its potential of wide applicability.

2 Cellular Network Primer

Figure 1 (left) illustrates a simplified cellular network architecture. The mobile device (*i.e.*, smartphone) connects to the Internet or telephony network through the base stations and the cellular core network. Both control-plane and data-plane (*i.e.*, user plane) operations are needed to receive data/voice services. The data plane delivers user content (data/voice), whereas the control plane exchanges signaling messages to facilitate content delivery.

Cellular network protocol stack. Figure 1 (middle) shows the cellular protocol stack at the device, which has three parts. The first is to enable radio access between the device and the base station.

Category	Description
Network status	get current radio state, signal strength, time changed
Network setting	barring, forwarding, selection ...
Call/SMS/Data	call status&handling (*e.g.*, dial, answer, mute), SMS, ...
Miscellaneous	power, reset, vendor-defined support

Table 2: Solicited commands supported by RIL (Android) [22].

Physical (L1) and link (L2) functionalities, including PHY, MAC, RLC (Radio Link Control) and PDCP (Packet Data Convergence Protocol), are implemented. The second part is the control-plane protocols, which are split into access stratum (AS) and non-access stratum (NAS). The AS protocols regulate radio access through Radio Resource Control (RRC). RRC is mainly for radio resource allocation and radio connection management; it also helps to transfer signaling messages over the air. NAS is responsible for conveying non-radio signaling messages between the device and the core network. Two protocols of mobility management (MM) and session management (SM) also belong to the control plane. MM offers location updates and mobility support for call/data sessions, while SM is to create and mandate voice calls and data sessions. The last piece is the data-plane protocols above IP, which are not cellular specific but use the standard TCP/IP suite.

Table 1 lists the protocols studied in this work. Multiple variants exist for a common function (like RRC and NAS) in 3G and 4G. 3G also has variants for its circuit-switched (CS) and packet-switched (PS) domains for voice and data, respectively. L1/L2 protocols and control-plane protocols are generally cellular specific.

Limited in-device access through APIs. In commodity phones, the OS and mobile apps have limited access to low-level, cellular-specific information at runtime. As shown in Figure 1 (right), cellular-specific protocols (say, control-plane and L1/L2 protocols) are implemented within the chipset (*e.g.*, Qualcomm Snapdragon and Samsung Exynos). As a result, cellular-specific information is mostly inaccessible to the software (for both kernel-space and user-space) in usual scenarios. The OS gets access to *basic* cellular functions and states (*e.g.*, registration, dialing a voice call and enabling/disabling data) through the *de facto* radio interface layer (RIL) library which interacts with the cellular interface exposed by the chipset. The RIL implementation is vendor-specific, and relies on the standardized `AT` commands [19]. For ease of app development and permission control, the OS further encapsulates a subset of RIL library to APIs, *e.g.*, `TelephonyManager` class for Android [6, 7]. Some system services on specific phone models (*e.g.*, FieldTestMode in Nexus 5 [20] and iPhone [21]) may directly access some, but not all cellular information from the RIL interface. Table 2 offers a sample of RIL commands [22], which exposes coarse-grained information (call/data/cell level) only.

Debugging tools, such as QXDM [2], XCAL [3], MTK Catcher [4], xgoldmon [23], can collect cellular network messages and offer fine-grained information. However, they all work with PCs, and do not offer in-device collection or protocol analytics (see §9 for more discussions).

3 MOBILEINSIGHT Overview

MOBILEINSIGHT offers a pure software-based solution for in-device collection and analytics of cellular protocol information. It runs as a user-space service. It infers protocol operations and key configurations by exploiting messages exchanged between the device and the network at the hardware chipset. It supports *fine-grained*, *per-message* information retrieval and analysis from a set of cellular-specific protocols on the control plane and at lower layers. It not only unveils *what* is going on with cellular-specific op-

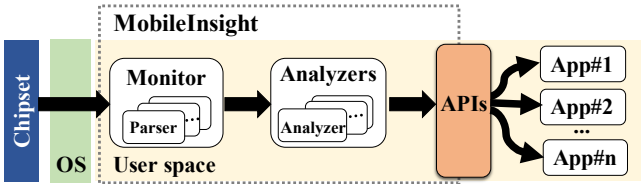


Figure 2: MOBILEINSIGHT architecture.

erations, but also sheds light on *why* and *how*. Specifically, MOBILEINSIGHT seeks to achieve three concrete goals.

- *In-device deployability*. It should be readily deployable in COTS phones without extra hardware or changes on the existing infrastructure or the device OS.
- *Protocol analytics.* In addition to archiving protocol messages, MOBILEINSIGHT should supplement analytics for standardized cellular protocols, including their state dynamics and operation logics. Ideally, the analysis is done at runtime, so that it can be used for various usages such as performance improvement and failure diagnosis.
- *Fine granularity and wide coverage.* It should provide fine-grained information to runtime protocol operations. Moreover, it should support protocols across layers and on both control and data planes.

Figure 2 illustrates the architecture of MOBILEINSIGHT, which has two main components.

(i) Monitor (§4). It first exposes raw cellular logs from the cellular interface to the device user-space at runtime, and then parses them into protocol messages and extracts their carried information elements. It builds an extensible modular framework, where each parser works on a per-protocol basis. The parsed messages are then fed to the analyzer.

(ii) Analyzer (§5). Given the extracted messages, the analyzer aims to unveil protocol dynamics and operation logics. Based on the observed messages and the anticipated behavior model (from cellular domain knowledge), the analyzer infers protocol states, triggering conditions for state transitions, and protocol's taken actions. Moreover, it infers certain protocol operation logics (say, handoff) that uses the operator-defined policies and configurations. It offers built-in abstraction per protocol and allows for mobile OS-/app developers to customize their analyzers.

4 In-Device Runtime Monitor

To enable in-device runtime monitoring, we need to address three issues: (1) How to expose raw cellular information from the hardware to the software (§4.1)? (2) How to decode the information into valid messages, given rich types and inter-dependency of protocol messages (§4.2)? (3) How to meet the requirement for low latency and reduce the system overhead (§4.3)? We next elaborate on each.

4.1 Exposing Raw Logs from Side Channel

The first issue is that ordinary in-device schemes cannot expose message-level cellular information to the user space (see §2). We thus leverage an alternative side channel between the chipset and the software. We find that the chipset supports an external diagnostic mode, which exposes the cellular interface to the USB port. In fact, this diagnostic mode exists for major cellular chipsets (including Qualcomm, MediaTek and Intel series) and mobile OSes (including Android and iOS). However, no public documents are available for this mode. We have to learn its details from the open-source code of diagnostic drivers (summarized in Table 3).

Mobile OS	Chipset	Virtual device	Driver code
Android	Qualcomm	/dev/diag	[25]
Android	MediaTek	/dev/ccci_md_log_ctrl	[26]
Android	Intel XMM	/dev/mdmTrace	[23]
iOS	Apple A6	/dev/tty.debug	[27]

Table 3: Summary of virtual devices for external diagnostic mode.

Figure 3a illustrates the USB-based diagnostic mode on Android for Qualcomm chipsets. The cellular interface maps itself to a virtual device (*e.g.*, `/dev/diag`) in the OS. Different from RIL, this virtual device exposes *all* raw cellular messages as binary streams. When the USB is connected to the external collector (*e.g.*, a PC), the OS uses USB tethering [24] to bind the virtual device with a USB port (*e.g.*, `/dev/ttyUSB`). The external collector thus fetches the cellular messages from the hardware interface. This is how the debuggers (Qualcomm QXDM [2], MediaTek Catcher [4], Intel xgoldmon [23], XCAL [3]) collect logs from USB. Similar virtual devices also exist on other chipsets and mobile OSes (summarized in Table 3), with slightly different implementations[2].

MOBILEINSIGHT emulates an external logger at the mobile device to collect raw cellular logs. We issue commands directly to the virtual device (*e.g.*, via `ioctl` or AT command `AT+TRACE`, depending on the chipset and mobile OS types). These commands include activation/deactivation of cellular message types, and callback registrations to receive hex logs. We then pull the hex log streams from the virtual device, and pass them to the in-device message parser. This ensures that, for each cellular message accessible to external debuggers, it is also available from MOBILEINSIGHT.

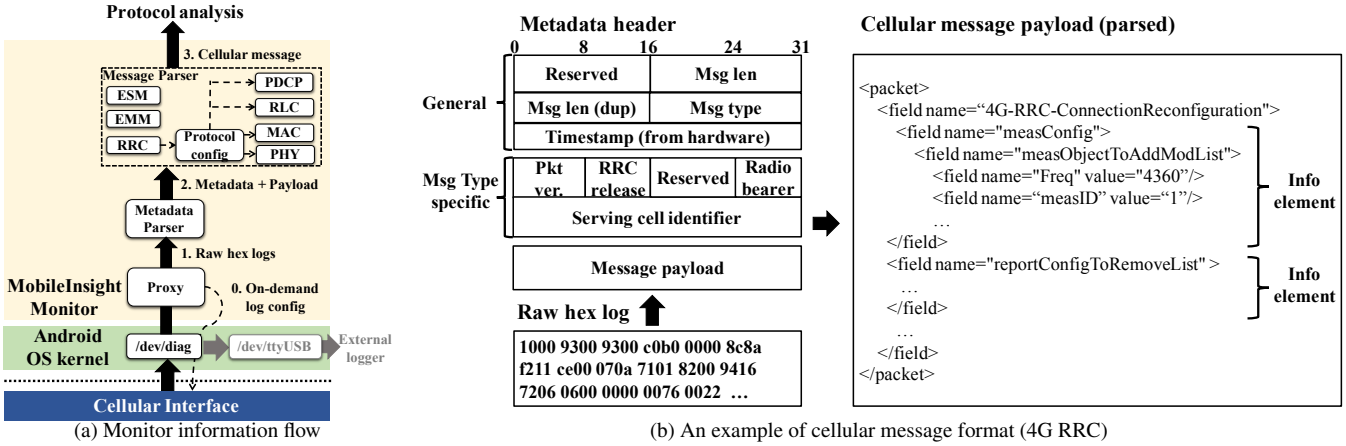
4.2 Parsing Cellular Network Messages

Given the raw cellular logs, we next parse each message. The issue is to decode a variety of message types (see Table 4) in their rich formats. Figure 3 exemplifies the structure of the 4G RRC message from the side channel. It carries a metadata header and the payload, plus configurable and message-specific information elements. The metadata headers' formats are specific to cellular chipsets. We infer them based on the raw binary logs from the diagnostic virtual device, and the publicly available open-source driver code [23, 25–27]. The message-specific information elements are standardized in the 3GPP standards [9–12, 16, 17, 17, 18, 28].

MOBILEINSIGHT parses such rich messages in two steps, as shown in Figure 3a. During the first step, a *metadata parser* is applied to the raw hex logs to extract the message type ID and release version. It then selects the corresponding *message parser* with a switch branch over the (type-ID, release) tuple. To develop message parser for each signaling message, we extract the message formats from the standards of each protocol. Some formats can be automatically extracted. For instance, the 3G/4G RRC standards provide abstract message notations under ASN.1 [29], which can be readily compiled into message decoders. For other messages, we manually convert them to machine-readable formats.

Handling protocol dependency. Certain messages are inter-dependent even at the parsing level. L1/L2 protocols (PDCP/RL-

[2]It is possible that some phone models do not have the virtual device. For example, we find several LG Nexus 5 and Motorola Nexus 6 models delete it on startup. In this case, neither external debuggers (*e.g.*, QXDM) nor MOBILEINSIGHT can extract runtime cellular function. MOBILEINSIGHT can detect the inaccessibility of the side-channel, and stop any tasks if it is inaccessible. In reality, we observe that for each phone model, the sidechannel accessibility is highly stable: it either always exists (for most phone models), or is removed by some phone vendors permanently.



(a) Monitor information flow

(b) An example of cellular message format (4G RRC)

Figure 3: MOBILEINSIGHT's in-device runtime monitor.

Protocol	Message Types	# Msg	# Element
4G-PHY	PDSCH signal; Cell Measurement	2[a]	N/A[b]
4G-MAC	Uplink/downlink transport blocks; MAC config; Buffer status report	5[a]	54
4G-RLC	Control/data packet data unit	2[a]	N/A[b]
4G-PDCP	Control packet data unit	1[a]	N/A[b]
4G-RRC	System info blocks; Connection setup/release/re-establish/reconfig; Handover command; Measurement control/report; Radio capability equerry; Paging; Security model command	45	185
3G-RRC	Same as 4G-RRC	32	108
3G-MM/GMM	Attach/detach; Authentication request/response; Location update; Security mode control; Identification request/response; Service request; Paging	41	63
4G-EMM	Same as 3G-MM/GMM	32	108
3G-CM/SM	Session (EPS bearer/PDP context) setup/modify/release; PDN connect/release/modify	58	54
4G-ESM	Same as 3G-CM/SM	22	71
CDMA/EvDo	Paging information; connectivity establishment/release; radio link protocol status	5[a]	N/A[b]

Table 4: Cellular messages in current MOBILEINSIGHT monitor.

[a] Not all the message types supported.

[b] No information elements are defined.

C/MAC/PHY) may need control parameters from RRC for correct parsing. For example, the PDCP packet headers might be compressed with RoHC [30], whose parameters are carried in the `RRC Reconfiguration` message. Without these RoHC parameters, these packets cannot be decompressed or decoded. We thus implement a protocol configuration repository. Upon receiving `RRC reconfiguration`, MOBILEINSIGHT extracts related information elements for PDCP/RLC/MAC/PHY, such as the compression parameters (for PDCP), acknowledgement mode configuration (for RLC), DRX timers (for MAC), modulation support (for PHY), *etc.* They are used to parse upcoming messages accordingly.

4.3 Optimization

We apply several optimizations to reduce message collection/processing latency and system overhead. First, MOBILEINSIGHT uses *on-demand collection* to only archive those logs required by the device-specified analyzers. It asks each protocol analyzer to declare its needed cellular messages (§5), and dynamically configures the cellular interface to record only those messages of interests. In §7.3, we will show that it can help reduce the storage overhead by up to two orders of magnitude. Second, it invokes *on-demand parsing* to only decode those necessary fields. For example, the analyzer may only want to learn the connectivity state in RRC. It thus parses the metadata only, and then passes the message to the analyzer with an annotation of the message parsers needed. Then the analyzer can parse it on demand by calling the `decode()`, which reads the annotation and calls the correct message parser. Last, we parallelize log collection and parsing. The trace collection proxy and the parser are two separate daemons, and the proxy passes the raw logs via an in-memory queue. This prevents that the analysis is blocked by log collection.

Table 4 summarizes the supported messages by the time of submission. For raw log collection, it supports the same types of messages as the state-of-art external debuggers. For message parsing, it currently supports 240 message types, encapsulated in 68 type-specific metadata headers and 3GPP releases 7-12. It decodes all signaling messages on radio resource control, mobility management and session management for 3G and 4G. It partially supports 4G PHY, MAC, RLC, and PDCP messages, mainly those conveying control information. It also supports CDMA/EvDO messages partially, including the paging and radio link protocols. We have realized full support for Qualcomm Snapdragon processors, and validated the feasibility on MediaTek/Intel chipsets and iOS.

5 Cellular Protocol Analytics

With the cellular messages, MOBILEINSIGHT further builds runtime analytics for protocol behaviors. Table 5 summarizes the protocol analytics we have developed. For each protocol, we uncover two dimensions of its behaviors:

- **Protocol state dynamics (§5.1):** They include the protocol states, and the state transition events. They are controlled by the standardized protocol state machines and runtime observation of protocol messages and configurations.
- **Protocol operation logic (§5.2):** It decides what parameters to use and which messages to send/receive. For network-centric 3G/4G design, it is the algorithm or policy used by the network operator to determine the parameters/messages used by protocols.

5.1 Extraction of Protocol State Dynamics

The cellular protocol states at the device are regulated by the state machine in 3G/4G networks. The runtime protocol state dynamics provide direct hints about performance (*e.g.* high/low-rate connectivity state in RRC) and functional correctness (*e.g.*, failure states

Analyzers	Description
LteRrcAnalyzer	4G RRC protocol analyzer that uncovers connection state, configuration dynamics and base station's handoff decision logic.
LteNasAnalyzer	4G EMM/ESM protocol analyzer that uncovers the mobility management state dynamics
LtePhyAnalyzer	4G physical layer analyzer that reveals the runtime radio resource allocation and link capacity
WcdmaRrcAnalyzer	3G RRC protocol analyzer that uncovers connection state, configuration dynamics and base station's handoff decision logic.
UmtsNasAnalyzer	3G GMM/SM protocol analyzer that uncovers the mobility management state dynamics

Table 5: Built-in protocol analyzers.

in mobility management or session management). For each signaling protocol (3G/4G RRC, MM and SM), MOBILEINSIGHT seeks to capture its *runtime* state dynamics, including the current state, the state transitions and the conditions for transitions.

We take a two-phase approach (see Figure 4 for an example). We first derive a reference state-machine model for each protocol based on the 3GPP standards [9–12]. This model abstracts the *device-side* states and transition conditions as a function of cellular messages. We then feed runtime cellular messages from our in-device monitor (§4) to this model. This provides the exact protocol states and state-related configurations. From those cellular messages, we derive the transition parameters and track the state transitions by following the reference state machine. Since both the standardized state machines and runtime messages are known in MOBILEINSIGHT, the *ground truth* on the runtime protocol states can be obtained.

Reference state machine. We focus on the protocol state machine at the device. For each protocol (RRC/MM/SM), the standard specifies the protocol states and substates, and the transition conditions. In RRC, the state represents the radio connectivity between the device and the base station. For MM, the state denotes the device's registration status to the core network. For SM, the state represents the data session activity and QoS configurations. Figure 4 exemplifies the 4G-RRC state machine defined in [12]. We extract both the main states (say, `RRC_IDLE` and `RRC_CONN`) and the substates at each state (*e.g.*, `Continuous-RX` and `Short/Long-DRX` in `RRC_CONN`). Each state transition is modeled as a boolean function of cellular messages: true if the transition condition is met upon receiving this cellular message. The transition can be directly triggered upon receiving certain messages, and/or controlled by parameters inside the message (say, timers). It can also be activated upon receiving multiple messages (*e.g.*, five rejection messages from MM lead to the "out-of-service" state [10]). Note that such a reference model itself does not provide runtime state dynamics or concrete parameter settings for the state transition. Instead, it serves as a template to track cellular messages and states at runtime. Table 6 summarizes the sizes of the device-side state machines for each protocol, which are independent of network carriers and phone models. It can be seen that all protocols' state machines are of modest sizes, thus able to be tracked efficiently at end device.

Runtime message-driven state tracking. Given the reference model, MOBILEINSIGHT next tracks state transitions based on incoming cellular messages from the in-device monitor (§4). It first reads the reference state machine, and determines the cellular messages to be monitored. Each observed message is passed to all those state-transition functions originated from the current protocol state. If any transition function is satisfied, MOBILEINSIGHT updates the current state. Figure 4 shows how this works for the 4G-RRC connectivity dynamics. Between idle and connected states, receiving the RRC Connection Setup/Release message immediately triggers the transition. The transitions between substates within `RRC_CONN` rely on timer configurations. In this case, the transition functions between substates extract timers from the runtime RRC Connection Reconfiguration message, and use internal timers to track the potential transitions. In the example, `CRX` switches to `Short-DRX` upon T_1 timeout, and moves back to `CRX` but not `Long-DRX` since timer T_2 stops. T_2 is configured one or multiple short DRX cycle (here, 2×20 ms = 40ms).

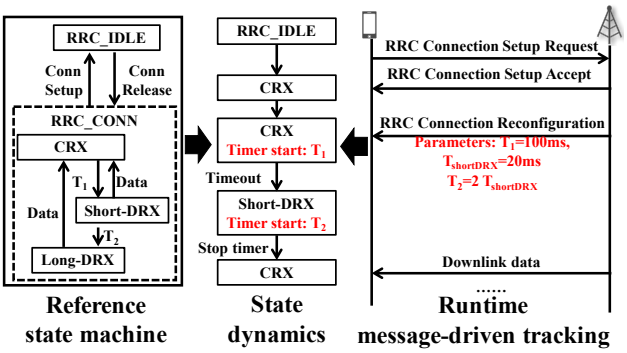


Figure 4: Example for 4G-RRC protocol state dynamics with reference state machine and runtime signals.

Protocol		#State	#State transition	Standard
Session management	CM	19	44	[9]
	SM	7	21	[9]
	ESM	4	12	[10]
Mobility management	MM	30	50	[9]
	GMM	18	22	[9]
	EMM	21	25	[10]
Radio resource control	3G-RRC	5	13	[11]
	4G-RRC	4	6	[12]

Table 6: Size of device-side 3G/4G protocol state machines.

Other protocols. We also apply similar techniques to track the states in 3G/4G MM and SM protocols (Table 6).

• **Mobility management:** These protocols control the device's registration status to the core network, and manage the tracking/location/routing area for the device. We track the device's registration status based on messages of attach/detach and location/routing/tracking area update. In each (de)registration status, the device's configurations (*e.g.*, security mode, voice usage preference, network features) are also recorded. Such information can be used for failure diagnosis and security loophole detection (see §8).

• **Session management:** These protocols, including 4G ESM, 3G CM/SM, control the device's data/voice session activities. Each data session has its own QoS profile (*e.g.*, traffic/delay class, maximum bitrate) and data billing policy. In MOBILEINSIGHT, we track the data session activity based on the session setup/modify/release messages. We extract the QoS profile and the billing policy (in the form of traffic flow template) from these messages, and use them as hints for data performance and network failure diagnosis (see §8).

Correctness. MOBILEINSIGHT provides protocol state dynamics identical to those constructed by using messages from external debuggers (*e.g.* QXDM and MTK Catcher). This is for two factors: (1) The protocol state machines are standardized, while the standards dictate the protocols to follow the runtime parameters; (2) MOBILEINSIGHT has access to the same cellular information as those tools. In the RRC protocol context, MOBILEINSIGHT directly extracts and predicts RRC states with *explicit* information. It is thus better than the implicit learning scheme (*e.g.*, through power measurement [31–33]). Similarly, MOBILEINSIGHT directly ana-

lyzes 4G EMM and ESM protocols, and their 3G variants as well. Note that, however, the ultimate correctness of our protocol state tracking at the device depends on two premises: (1) the device chipset implementation follows the 3GPP standards; (2) the cellular information from the diagnostic mode is accurate. We do not have any evidence to show that neither premise is invalid now.

Limitations. While MOBILEINSIGHT can accurately reconstruct the device-side protocol state dynamics, it does not have direct access to the network-side protocol state counterparts (which can be different from client-side protocol state dynamics). Indeed, the network-side protocol state dynamics could be inferred based on the device-side protocol's state, which has been regulated by the 3GPP standards. We leave this inference to the future work.

5.2 Inference of Protocol Operation Logic

MOBILEINSIGHT can also infer certain protocol operation logic from the network. The logic is the algorithm or policy by the operator to determine what configurations the protocol should use and what messages to send/receive. By analyzing operation logic, the device can forecast possible performance degradation (*e.g.*, handoff to a low-speed cell) and functional incorrectness (*e.g.*, network failures). We next present our initial effort on inferring the network-side protocol logic using the case study on handoff.

Handoff switches the device's serving cell from one to another. It is critical to end devices, since the target cell to be chosen may have varying performance. In 3G/4G, when the device is at `RRC_CONN` state, the handoff decision is made by the base station and assisted by the device. Figure 5 (top) depicts a typical handoff procedure and its signaling between the device and the network. The phone is initially served by BS1. The serving cell (BS1) asks the phone to measure and report the radio quality of neighboring cells. Upon receiving the measurement report from the phone, BS1 runs its decision logic to determine whether handoff should be triggered. It may reconfigure the device for further measurements (right), or issue the device handoff command (left).

We need to address two challenges when inferring network-side handoff logic. First, the connected-state handoff decision logic can be operator specific. The 3GPP standards leave the freedom for operators to customize their decision logic. Second, the device does not have full access to all network-side operations. It has to rely on its observations and interactions with the network to learn the logic.

Fortunately, the operation logic is not arbitrary in reality. It typically follows well-justified common practices [34–40]. Operators tend to apply the stable logic to each cell. It remains stable, and thus predictable in operational 3G/4G networks. Moreover, we observe that *many network-side protocol operations be interactive and stateful*. The logic is customizable, but regulated by standardized mechanisms at the protocol level. The network often relies on device feedback to operate its protocols (*e.g.*, measurement report for handoff). Consider the example of Figure 5. To make a proper handoff decision, the serving cell needs to know the *device-perceived* signal strength of nearby candidate cells. It thus configures the device to perform measurements and report the signal strength (via the `Meas Control` command). This interaction may take multiple rounds, because the base station may request the device to measure more candidates based on prior measurements. According to 3GPP standards [11, 12], both the handoff commands and measurement report criteria are of *limited* options. Although the device has no direct access to network-side operations, it may infer them by *pairing control commands and feedbacks*.

Consequently, we model network operations as a finite-state machine and devise an online inference algorithm. Our approach

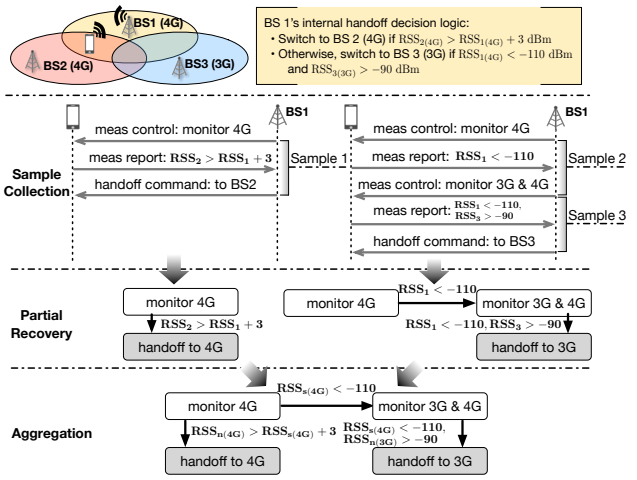


Figure 5: An example of 4G cell handoff decision logic (top), and how it is inferred by device at runtime (bottom).

adapts QSM, a state merging algorithm [41, 42] in AI with leveraging domain-specific knowledge on cellular networks to improve inference accuracy.

Modeling handoff decision logic. We model the handoff logic as a domain-specific finite-state machine. Our model takes into account the standardized mechanisms, including measurement control, measurement report and handoff procedure. Each state denotes the *device's* control state configured by the network (*e.g.*, `Meas Control` and `Handoff Command`). Two states are *equivalent* if their control parameters are identical. For `Meas Control`, this means identical measurement report criteria (*e.g.*, A3 defined in [11, 12]). For `Handoff Command`, we assume that they are equivalent if the target cells are identical. The state equivalence is essential for the state merging process. The state transition happens when a new control command (`Meas Control` or `Handoff Command`) is received by the device. It is invoked upon receiving the device's `Meas Report` message in response to the current control state. Following QSM, the state transition is modeled as a *prefix* of `Meas Report` sequence. Any sequence matching this prefix would trigger the state transition. This model is valid for handoff, because the base station may make handoff decisions before receiving all reports from the device.

Online inference. We use an online algorithm to infer the handoff decision logic. For each serving cell, it collects runtime handoff events and associated measurement controls/reports as samples, and *iteratively* updates its inferred state machine by aggregating a new state with the existing one. Each iteration has three steps: *sample collection*, *partial recovery*, and *aggregation*.

(a) Sample collection. We collect the training samples at runtime without active probing of the cellular network. We define a *sample sequence* as the tuple of an old control command (*e.g.*, `Meas control`), a new control command (*e.g.*, `Meas control` or `Handoff command`) and the `Meas Report` sequence in between. To collect a sample sequence, we track all corresponding messages in the background (via in-device monitor) until the next control command arrives. In the example, three samples are collected in two cases where the device hands over from 4G BS1 to 4G BS2 (left) and 3G BS3 (right).

(b) Partial recovery. From each sample sequence, we generate a state transition by converting the old/new control command into from/to the control state, with the feedback se-

Algorithm 1 G = Aggregation(G, Ξ)

Input: G =FSM of handoff logic, $\Xi = \{e\}$ with all transitions in (b);
for $e(v_s \xrightarrow{r} v_d) \in \Xi$ **do**
  if $v_s \in G$ AND $v_d \in G$ **then**
    Find $e' \in G$ satisfying $v_s \xrightarrow{r'} v_d$ and update $r' = r' \cup r$;
  else if $v_s \notin G$ AND $v_d \notin G$ **then**
    $G_e = \{e\}, G = G \cup G_e$; *//e is the only edge in an isolated* G_e
  else // v_s or $v_d \in G$, assume $v_s \in G$ for simplicity
    $G = G \cup e$ by adding v_d and the edge e into G
  end if
end for
Return G

quence being the transition condition (the sequence itself is also a prefix). Use sample sequence 1 as an example: `Meas Control` $\xrightarrow{\texttt{MeasReport1,...}}$ `Handoff Command`. We thus derive a transition between the state *"monitor other 4G neighbor cells"*→*"handoff to another 4G cell"* (here, BS2) when the measurement report indicates $RSS_2 > RSS_1 + 3$ (event A3 in [12]). Similarly, we derive the transition of extending the measurement from 4G to 4G and 3G, and the transition to a 4G→ 3G handoff.

(c) Aggregation. When a new partial transition is created, the aggregation step merges it to the existing state machine. It works in three steps. First, it performs symbolic mapping to generalize the rule. This is feasible because the 3GPP standards define the measurement control/report parameters in an abstract form [12]. For example, we translate $RSS_2 > RSS_1 + 3$ into a general rule $RSS_{n(4G)} > RSS_{s(4G)} + 3$ by mapping cells 1 and 2 into their roles in 4G: the serving cell and the neighboring candidate. Second, it locates where to merge. For each edge from the partial transition sample, we search if it exists in the current state machine. We use a directed acyclic graph G to represent the state machine. Finally, we merge the new rule into the existing graph by running the union operation over the graph.

Algorithm 1 shows the pseudo-code for aggregation. There are three cases: both source and destination states (nodes) exist in the graph, only one node exists, and no nodes exist. If no nodes are not found, it is treated as a new edge and added to the existing state machine as an isolated graph. When only one node exists, we create a new edge from the existing graph (by adding the non-existing node) and initialize its transition condition as the measurement sequence from the sample. When both nodes exist, the transition condition (prefixes) should be merged. We search for the longest common prefix in the existing transition and merge it with this new prefix. In theory, the old and new conditions for the same transition might differ or even conflict with each other (*e.g.*, $RSS > -110$ and $RSS < -110$). However, it would not occur in practice because the rules used by the same serving cell are consistent.

Ideally, the above algorithm should be performed over *every serving cell*. In practice, however, per-cell inference suffers from insufficient training samples; otherwise, the user has to wander around to collect unique samples within one cell coverage. Moreover, it requires more storage for per-cell logic. To tackle this issue, we observe that operators tend to apply the same logic to each cell type (*e.g.*, under the same frequency), with minor tuning on parameters (*e.g.*, thresholds). It is thus feasible to aggregate samples from cells of the same type, and infer the decision logic from each frequency. The above aggregation algorithm still applies. We merge those samples from the serving cells over the same frequency. Though this may lead to conflicting decision logics between cells of the same type in theory, it is unlikely to happen in operational 3G/4G network (§7.2). To handle it, we duplicate the state transitions and create two branches and mark it as an exception for further checking.

Correctness and limitations. The correctness of the inferred handoff logic is generally ensured by the good properties of QSM. [41–44] prove that, the state machine can be fully recovered if sufficient samples can reach all states and differentiate any pair of non-equivalent states in the logic. This requires the device to receive all possible types of *Meas Control* and *Handoff Command* from sufficient samples. If the device has not collected enough samples that meet above conditions, the state machine may be incomplete.

A limitation of our approach is that, the inference may not capture the internal states on the network-side handoff logic that do not interact with the device. For instance, even when the radio-related handoff criterion is met, the network may not invoke the handoff to the target cell for load balancing; such operations may remain invisible to the device. The inferred one would then be the *device-perceived* handoff logic only. The direct access to these network-side internal states would be possible only if the cellular infrastructure is open, which however is unlikely to occur in reality. Our study shows that our inference typically captures network-side logic in practice (§7.2).

Operation logic for other protocols. Besides handoff, other cellular protocols may also use their own logic. For example, operators may customize their QoS allocation policies in SM. PHY may customize its radio block allocation and rate adaptation algorithms. We are exploring to adapt our online inference to these contexts.

6 Implementation

MOBILEINSIGHT seeks to provide an open platform to facilitate researchers and developers to learn the protocol operations in cellular networks. It thus defines simple APIs for its monitor and analyzer components. We implement MOBILEINSIGHT on off-the-shelf smartphones, as a user-space service. We choose the user-space rather than in-kernel solution for ease of deployability. It consists of 29,698 lines of code (12,254 lines of C/C++ and 17,444 lines of Python), excluding the 3rd-party libraries. Our current implementation is mainly on Android phones with Qualcomm chipsets, but porting to other platforms is ongoing.

MOBILEINSIGHT API. We first illustrate how to use the API via an example (more detailed usage can be found in [45]), which seeks to analyze the protocol state dynamics of 3G and 4G RRC.

```
# Initialize a in-device monitor
src = Monitor()
#Declare 3G/4G RRC analyzers
lte_rrc_analyzer = LteRrcAnalyzer() #4G RRC
wdcma_rrc_analyzer = WcdmaRrcAnalyzer() #3G RRC
#Bind the analyzers to the monitor
lte_rrc_analyzer.set_source(src)
wcdma_rrc_analyzer.set_source(src)
#Start processing
src.run()
```

Both monitor and analyzer functions in MOBILEINSIGHT are encapsulated into classes, for instance, `Monitor` class and `LteRrcAnalyzer` class. MOBILEINSIGHT abstracts the inference per protocol into a module called *analyzer*. To call a chosen function, an atop app/service has to initiate an instance of its corresponding class. For example, `src = Monitor()` creates a `Monitor` instance. Second, the target app/service declares the needed analyzers, and binds them to the monitor via `set_source(src)` method. This lets an analyzer register a callback function upon certain cellular events from moni-

tor. Finally, we start MOBILEINSIGHT by running the monitor via `src.run()`, which logs the events and drives the analysis.

In-device monitor (§4). We implement the monitor using two daemons: a proxy daemon to extract raw hex logs from the cellular interface (chipset), and a parser daemon to decode messages. This allows for pipeline parallelism, thus reducing processing latency. The proxy daemon retrieves raw logs by leveraging Android's open-source driver for the cellular virtual interface [25]. Specifically, we first open the virtual device `/dev/diag`, and enable the logging mode by sending a command (defined in [25]) via `ioctl` function. We then register a callback function linked to the virtual device to be notified whenever raw binaries are generated. The parser daemon implements the decoding of cellular messages in Table 4. We also implement optimization techniques in §4.3, including on-demand collection, on-demand decoding and in-memory processing. To further speed up processing, both daemons are implemented in C/C++ and compiled with Android NDK.

Built-in Analyzers (§5). We implement each built-in analyzer as a Python module[3]. We port the Python-based analyzer framework via `python-for-android` [46] which allows to compile the Python code into Android apk. The analyzer framework integrates all analyzers into a directed acyclic graph. Each node is an analyzer, and a directed edge v→w denoting the dependency of w on v. Each analyzer is initiated at most once. It is shared by multiple callers when needed.

Miscellaneous issues. We discuss two related issues.

∘ *Message coverage.* The current version has not covered all cellular messages to date. We only focus on those most useful ones (control-plane and L1/L2 ones carrying control information). In principle, the same method is applicable to support all messages. We are extending MOBILEINSIGHT to data-plane protocols (below-IP) and their analysis.

∘ *Rooted phones.* MOBILEINSIGHT currently works with rooted phones. Studies claim that about 27.4% users have rooted their phones [47]. Root should not be a big problem at current stage. MOBILEINSIGHT's current target is the research community who does research on cellular networks. In fact, MOBILEINSIGHT only requires access permission to a specific system folder and the cellular interface. Once it is granted, it does not require other permissions for root privilege. To support more mobile devices, we are also exploring rootless techniques, such as building MOBILEINSIGHT as a system service, and customizing boot image with minimal modification to grant cellular access privilege to MobileInsight .

7 Evaluation

We next assess MOBILEINSIGHT to show its (1) in-device support for diverse phone models, chipsets and operators and wide-coverage of cellular messages, (2) effectiveness and runtime support, and (3) tolerable system overhead.

7.1 In-device Support and Wide Coverage

In-device support. We first validate that MOBILEINSIGHT is readily deployable over various phone models, mobile OSes, and cellular chipsets. We have installed and tested it over 30 phones. It works over all phones. Table 7 summarizes the phone models, covering a variety of Qualcomm chipsets and Android OS versions. We have also validated the feasibility of porting MOBILEINSIGHT to Intel/Madietek chipsets and Apple chipsets and iOS. Similar cellular messages are collected. We are extending our code to these phones. MOBILEINSIGHT works well with various operators. We have run experiments with 8 carriers: four US carriers (AT&T, Verizon, T-Mobile and Sprint), a US virtual operator (Project Fi) and three Chinese carriers (China Mobile/Telecom/Unicom).

Model	CPU	RAM	Chipset	OS
Huawei Nexus 6P	Quad-core 2GHz + Quad-core 1.55GHz	3GB	Snapdragon 810	Android 6.0.1
ZTE Nubia Z9	Quad-core 1.5 GHz	4GB	810	5.0.2
Motorola Nexus 6	Quad-core 2.7GHz	3GB	805	5.1.1
Samsung S5	Quad-core 2.5GHz	2GB	801	4.4.2
Sony Xperia Z3	Quad-core 2.5 GHz	3GB	801	4.4.4
Xiaomi Mi 4	Quad-core 2.5 GHz	3GB	801	4.4.3
LG G3	Quad-core 2.5 GHz	2GB	801	4.4.2
Samsung Galaxy Note 3	Quad-core 2.3 GHz	3GB	800	4.3
LG Tribute	Quad-core 1.2 GHz	1GB	400	4.4.2
Meizu MX4*	Quad-core 2.2 GHz	2GB	MediaTek MT6595	4.4.4
LG G4 Stylus*	Octa-core 1.4 GHz	2GB	MediaTek MT6592	5.1
Asus Zenfone 2E*	Dual-core 1.6 GHz	1GB	Intel XMM7160	5.0
Apple iPhone 5*	Dual-core 1.3 GHz	1GB	Apple A6	iOS 7

Table 7: MOBILEINSIGHT can run over various phones.

Wide coverage and characteristics of cellular messages. We validate that MOBILEINSIGHT supports a wide range of signaling messages and those L1/L2 ones conveying control information. We conduct both controlled experiments and a small-scale user study during an 11-month period (July 2015 - Jun 2016). In the controlled experiments, we test MOBILEINSIGHT with representative usage scenarios: static, walking and driving (local and highway) under various traffic loads – idle, voice, data with different rates. In the static test, we use iperf to generate constant UDP traffic. In the user study, 30 participating phones occasionally collect logs in the wild over 8 operators, with the total volume being 227.95 GB.

We next characterize their traffic patterns through the controlled experiments. Figure 6 shows illustrative traces collected by Samsung S5 over T-Mobile; other phones and operators have similar behaviors. We make three observations. First, the heavier data traffic result in more cellular messages (top two plots for static tests). The reason is that, more data delivery requires more control support and triggers more radio link reconfigurations. Second, mobility causes more signaling and dynamics. The third plot shows a driving test which starts around the 120th second (2nd minute), turns on mobile data (background) at the 600th second, and adds ping around the 1200th second. Mobility triggers frequent radio link reconfigurations at PHY. It also incurs more signaling messages from RRC and NAS. Peaks are observed for certain control events (handoff). Third, the lower-layer messages are much heavier than the higher-layer ones. This is because the device interacts with the core network much less frequently than with the base station. We show the message breakdown in the bottom plot ([1200, 1800] seconds of the 3rd plot) and the CDF in Figure 7a. PHY control messages are an-order-magnitude higher than MAC, which has another order-of-magnitude higher than RRC and NAS. The control plane (RRC and NAS) messages yield a bursty pattern. They come every several seconds, but can reach up to 20-30 messages/s. Note that MOBILEINSIGHT has not supported a full set of L1/L2 cellular messages (see Table 4) and the total traffic is underestimated.

Figure 7b shows he statistics from the user-study dataset. We find that the results are device independent and combine them in analysis. There are much more 4G messages (91%) than 3G (9%). This indicates the popularity and wide deployment of 4G. We ob-

[3]We choose Python instead of Java-based Android programming for two reasons: (1) cross-platform support: MOBILEINSIGHT has both in-device and desktop versions on Windows/Linux/OS X. Using Python ensures that the analyzers can run on all platforms without modifications; (2) extensibility: MOBILEINSIGHT aims to support direct execution of analytics as plugins. Python is more suitable for this purpose.

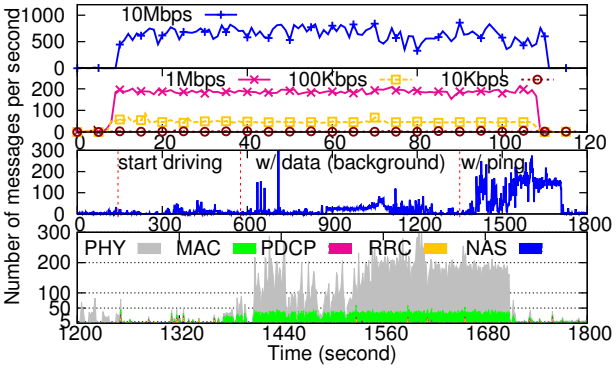
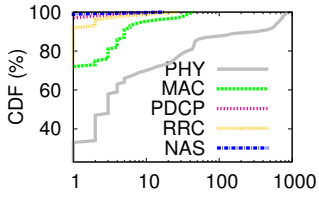


Figure 6: Cellular message traffic patterns in the controlled tests.



(a) CDF in a driving test

# volume	227.95 GB
# msg	67,285,683
4G/3G	4G (91%), 3G (9%)
Release	Rel-7, Rel-8, Rel-9 , Rel-10, Rel-11, Rel-12
Layers	PHY (71.8%), MAC (9.0%), PDCP (8.3%), RRC (10.0%), NAS (0.6%), CDMA/EvDo (0.3%)

(b) User-study dataset statistics

Figure 7: Cellular traffic characteristics.

serve control information from various 3GPP releases (from Rel-7 to Rel-12). This implies the hybrid deployment and evolution of the infrastructure. Moreover, NAS messages are much fewer than RRC and PHY. This is also because the device interacts with the core network much less frequently than with the base station. CDMA/EvDo messages are also observed in Verizon and Sprint 3G. They are less observed because of our recent support for them and thus relatively less logs collected.

7.2 Responsiveness and Effectiveness

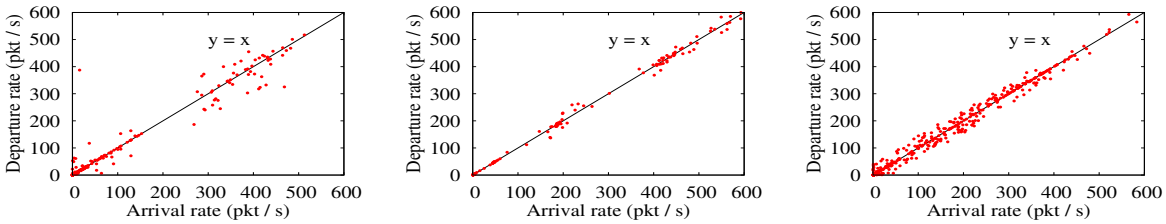
Message monitoring rate. We examine how responsive MOBILEINSIGHT parses runtime cellular message. We record every cellular message's arrival (at monitor) and departure (after being parsed) timestamps, and calculate the message number every second. Ideally to satisfy the realtime requirement, the departure rate should match with the arrival rate. This corresponds to a line $y = x$, with x as the arrival (generation) rate from the hardware interface, and y as the departure rate from MOBILEINSIGHT's monitor. In this experiment (and hereafter), we mainly use three phone models with different capabilities: low-end (LG-Tribute), medium-end (Samsung S5) and high-end (Nexus 6P) in this test (summarized in Table 7). Figure 8 shows the result using AT&T. The results are similar for other operators. The results show that MOBILEINSIGHT approximates the ideal line $y = x$ for most phones in most cases. Only for the low-end one, the processing speed slightly vibrates around the line. It implies that the monitoring sometimes (mainly under heavy load) lags a little bit behinds but is compensated afterwards (see the points below the curve). It does not hurt the monitoring because two separate processes are used for proxy and parse daemons (§4.3).

Processing time. We examine how responsive MOBILEINSIGHT parses and analyzes messages. For each message, the *processing time* is defined as the elapsed interval from its arrival to its completion of all analyses. In this experiment, we run the in-device logger with all supported signaling messages activated (Table 4), and all protocol analyzers enabled (Table 5). Figure 9 shows the processing time under light ($<$ 500 msg/s) and heavy ($\geq$ 500msg/s) loads. MOBILEINSIGHT completes 99% processing in less than 0.8ms, except the low-end, heavy-load case that it finishes 90% within 0.8ms. The maximum processing time observed is 33ms. The low-end one (1.2GHz CPU) performs slightly worse than the other phones. This validates MOBILEINSIGHT largely meets the realtime requirement.

Effectiveness in extracting protocol state dynamics. MOBILEINSIGHT can correctly track runtime protocol states and key configurations. For each signaling protocol, we compare the signaling messages from MOBILEINSIGHT and those from QXDM. We confirm that they are identical, because their data sources are the same. We use the logs from our user study to retrieve the protocol state machines. We find that all follow the standard specifications, and no mismatch is observed. All protocol states summarized in Table 6 can be observed under different scenarios. The elapsed time needed by MOBILEINSIGHT to track the current protocol state is bounded by the message processing time, which is less than 0.8ms for 90% messages and 33ms at maximum (Figure 9).

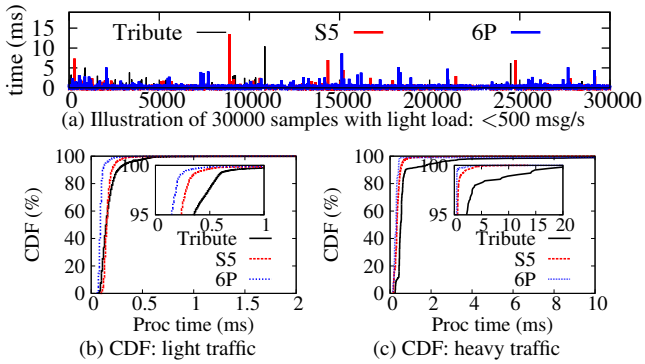
We summarize some key 4G protocol configurations retrieved from our user study in Table 8. These parameters unveil essential runtime information for end devices, including RRC connectivity timers (§5.1), ciphering/integrity protection parameters from mobility management protocol, and QoS profiles from the session management protocol. We clearly observe diversity among carriers and even within each carrier. For example, Sprint chooses different RRC timers from others. In most cases, it does not enable short DRX (0 means no T_2 and a direct jump to long DRX). Its T_1 uses three options (10/100/200 ms); Its short DRX cycle is mainly configured as 80ms (40ms in few samples, 0.3%). We have also seen diversity in encryption and QoS settings. Note that, China mobile does not enable encryption, thus exposing itself to attacks (see the security followup result in §8). EEA (EPS Encryption Algorithm) and EIA (EPS Integrity Algorithm) are the standardized cipher algorithms for 4G LTE. For QoS, the larger the value, the lower the QoS. It can be used to troubleshoot performance issues (§8).

Effectiveness in inferring operation logic. We next show that MOBILEINSIGHT can infer most handoff operation logics defined by network operators. We first infer handoff policies per cell and then assemble them per frequency. Figure 10 shows four 4G instances inferred by MOBILEINSIGHT (one for each US carrier). A frequency band unit is indexed by an EARFCN (E-UTRA Absolute Radio Frequency Channel Number) [48]. Each carrier has multiple licensed channels for 4G and 3G. For example, channels 5780, 1975, and 825 are three LTE ones used by AT&T, with downlink center frequencies at 739MHz, 2112.5MHz and 1952.5MHz. Due to space limit, we do not show the handoff policies inferred for all channels. They are in a similar form, but parameters and states may differ. For instance, 3G→4G handoff has distinct rules from the 4G→3G one. Our inference process validates the hypothesis that operators do follow well-justified practices [34–36]. All operators almost use the identical handoff policy per frequency. So each cell's state machine over the same channel can be readily aggregated without conflicts; Each observation either coincides or complements another. The aggregated handoff logic merges quickly within several rounds. It also implies that, the inferred handoff logic is likely to be correct because it is consistent among the same type of cells. We also observe that intra-frequency handoff (over the same frequency) is the most common one with the highest priority. The handoff logic varies with carriers. Indeed, operators have freedom to customize their policies. We have also applied the



(a) LG Tribute (b) Samsung Galaxy S5 (c) Huawei Nexus 6P

Figure 8: MOBILEINSIGHT message departure rate as a function of arrival rate from hardware cellular interface.



(a) Illustration of 30000 samples with light load: <500 msg/s

(b) CDF: light traffic (c) CDF: heavy traffic

Figure 9: MOBILEINSIGHT's processing time under light (<500 msg/s) and heavy (≥500 msg/s) loads.

		AT&T	T-Mobile	Sprint	Verizon	CMCC
RRC	T_1 (ms)	200 (99.9%) 100 (0.1%)	500 (100%)	200 (54.4%) 100 (31.2%) 10 (14.4%)	200 (99.5%) 10 (0.5%)	60 (100%)
	$T_{shortDRX}$ (ms)	20	80	0 (85.6%), 80	40	20
	$T_2/T_{shortDRX}$	1	1	0 (85.6%), 2	0, 4	2
	Conn→ idle (s)	10 ±0.9	10 ±0.6	4 ±0.5	10 ±0.8	13 ±0.5
EMM	Encryption	EEA2	EEA2	EEA2	EEA2	**NULL**
	Integrity	EIA2	EIA2	EIA2	EIA2	EIA2
ESM	QoS class	8	6	9	1 (voice), 5	9
	Max dlink bitrate	150Mbps	256Mbps	200Mbps	138Mbps (data), 80Kbps (voice)	256Mbps

Table 8: 4G protocol state-relevant parameters retrieved in 5 carriers.

same inference to AT&T and T-Mobile 3G, and verified that similar per-frequency aggregations are used. One major difference is that, only intra-frequency handoff is allowed for each 3G frequency band. This is possibly because that 3G supports soft handoff between cells under same frequency. So intra-frequency handoff is more preferred to provide more seamless network service.

Given that carriers do not publicize their operation logics, we lack the ground truth to directly assess the correctness of our inference algorithm. We thus devise an indirect approach. The idea is to predict whether a handoff indeed occurs based on the inferred handoff logic. If our inferred state machine is entirely identical to the one used by the operator, we could predict the handoff without errors. The accuracy reflects the completeness and correctness of prediction. Note that, the inferred one may be incomplete due to device-based perception only. In a word, the prediction accuracy serves as a lower bound of the inference correctness. We run the 10-fold cross validation [49]. Table 9 shows that MOBILEINSIGHT yields high prediction accuracy (87.5% to 95.3%) for four US carriers. It is slightly lower for Verizon because of a relatively smaller dataset. This validates that in practice, MOBILEINSIGHT is effective in learning the network-side operation logic. There are only false positive errors (*i.e.* the handoff does not occur when the radio measurement meets the trigger conditions). No false negative errors are observed since the handoff decision indeed considers radio quality [36]. These results confirm that, our inference is almost complete, covering most rules used by operators through the device-only observations.

	AT&T	T-Mobile	Sprint	Verizon
#Samples	11,050	10,178	10,042	2,741
Accuracy	90.7%	91.8%	95.3%	87.5%

Table 9: Accuracy for predicting upcoming handoffs.

	Control only	Control+data	Control+data+PHY
Hex log size (1-hour)	0.36MB	3.89MB	41.40MB
Avg. log growth speed	0.79Kbps	8.64Kbps	92.00Kbps
Avg. log size reduction	115.6x	10.62x	N/A

Table 11: Log storage overhead in MOBILEINSIGHT.

7.3 System Overhead

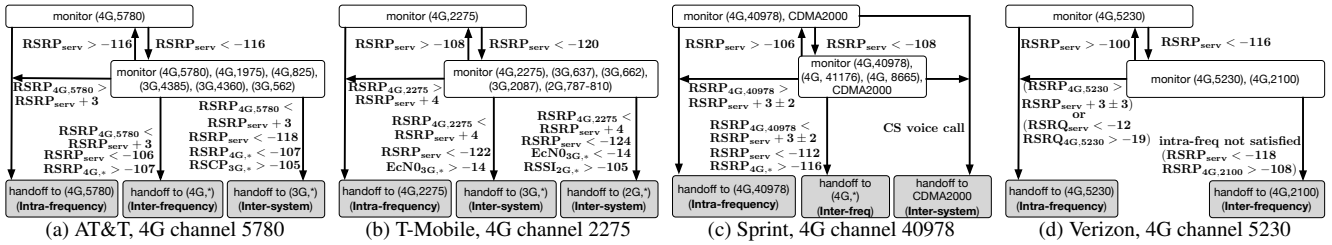
CPU and memory. We assess MOBILEINSIGHT's CPU and RAM usage. We enable all the supported messages (Table 4) and protocol analyzers (Table 5), and record their usage every second when we export parsed messages and analysis results into a file. We divide the resource consumption into two parts: by MOBILEINSIGHT service and by other relevant operations (*e.g.*, file I/O). Figure 11 shows the average CPU and memory usage. Regarding MOBILEINSIGHT itself, the CPU consumption for monitor and analyzer modules is about 1-3% for S5 and 6P, and 6-7% for LG Tribute. For all phone models, the average RAM usage is below 15 MB (maximum < 30MB).

Energy consumption. We assess the energy cost by measuring the phone's power consumption with/without MOBILEINSIGHT, through a Monsoon power meter [50]. We use only Samsung S5 because their back-covers are easy to remove for power measurement. Due to space limit, we show the results in three typical scenarios in Table 10. The results in many other scenarios are similar. On average, MOBILEINSIGHT consumes 11-58 mW extra power. The higher the traffic load, the larger the power value. However, its relative ratio remains within 4%, regardless of traffic variations.

Storage overhead. We last gauge the storage required for MOBILEINSIGHT cellular logs. We run MOBILEINSIGHT on Samsung S5 phone in idle mode for 1-hour, and record the raw log size and growth speed. We repeat this experiment by enabling different levels of cellular message types. Table 11 shows that, the log growth and storage consumption depends on the cellular message types to be collected. When only the control plane messages (3G/4G-RRC/MM/SM)are enabled, the log growth speed is 0.79Kbps on average, which is 115.6x slower than the scenario when all messages are enabled. Note that the data-plane and PHY-layer messages grow in proportion to the mobile user data volume. With active user data, the log size would grow even faster. This justifies our optimization of on-demand log collection (§4.3), which can help reduce the storage overhead based on app demands.

8 Showcase Applications

We now present three showcase examples to illustrate how MOBILEINSIGHT can assist in failure diagnosis, performance improve-



(a) AT&T, 4G channel 5780 (b) T-Mobile, 4G channel 2275 (c) Sprint, 4G channel 40978 (d) Verizon, 4G channel 5230

Figure 10: Four instances of device-experienced mobility management policies inferred from MOBILEINSIGHT.

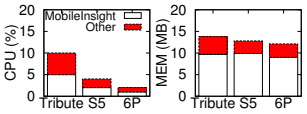


Figure 11: CPU & memory usage.

	idle	10Kbps	10Mbps
off	468 ± 15.0	824 ± 14.2	1792 ± 31.4
on	484 ± 17.1	835 ± 20.1	1850 ± 16.1
Δ	3.4%	1.3%	3.2%

Table 10: Consumed power (avg ± std) when MOBILEINSIGHT off/on.

Timestamp	Protocol	Event
17:57:24.814	3G-SM	PDP context setup request: QoS class = 1 (**voice**)
17:57:24.933	3G-SM	PDP context setup reject: QoS unsupported
17:57:25.435	3G-SM	PDP context setup request: QoS class = 1 (**voice**)
17:57:24.515	3G-SM	PDP context setup reject: QoS unsupported
...	3G-SM	...

Table 12: A network failure due to voice QoS misconfigurations.

ment, and security loophole detection. These examples are not intended to be complete. Instead, they aim to demonstrate the feasibility and usefulness of building apps with MOBILEINSIGHT. Moreover, MOBILEINSIGHT is not restricted to these example scenarios; other services/applications that gains from low-level cellular information may benefit from MOBILEINSIGHT.

Network failure resolution. The first example is `NetDiag`, a control-plane diagnosis tool built on top of MOBILEINSIGHT. In presence of network failures, `NetDiag` aims to provide end users reasoning about why they occur. By tracking control-plane protocol state dynamics and operation logic at runtime, MOBILEINSIGHT provides direct hints for network failures and helps to resolve them. `NetDiag` uses four built-in analyzers (Table 5): `LteRrcAnalyzer` and `WcdmaRrcAnalyzer`, `LteNasAnalyzer` and `UmtsNasAnalyzer`. It keeps track of the runtime protocol states for each protocol (3G/4G RRC, MM and SM). When an error state (*e.g.*, RRC connection disruption, MM "out-of-service" state, SM "data session deactivated") is traversed, it traces back all cellular events that lead to the transition to this error state from the initial protocol state, and reports the entire event sequence to the device. By examining the event sequence, MOBILEINSIGHT provides operation logs on why the error is triggered, and offers suggestions for the device-side fix.

We show how `NetDiag` helps to identify two real-world failure cases that have not been reported before. The first one arises from the device-side QoS misconfiguration of voice over LTE (VoLTE). Table 12 shows the traceback logs in 3G session management layer from `NetDiag`. It is observed on a Samsung S5 phone over T-Mobile, with the VoLTE capability enabled. The failure occurs when the phone migrates to 3G and attempts to activate its data session (*i.e.*, the PDP context). The device keeps on receiving rejected session requests from the network, and is stuck at the inactivity state of the 3G-SM protocol. By tracing the log, we find that the problem results from the QoS specified by the device in the request message. It requires the QoS profile optimized for the low-latency voice service. This request is accepted by T-Mobile 4G, but is rejected by T-Mobile 3G (probably due to network resource constraints). This is caused by the phone's problematic implementation. To support VoLTE, it improperly uses the low-latency QoS profile in each request, even using 3G. Given this hint, we have fixed this problem at the device by disabling the VoLTE feature when the device is in 3G.

Timestamp	Protocol	Event
15:20:34.525	3G-SM	PDP context setup request
15:20:35.633	3G-SM	PDP context setup accept: dlink peak tput=**2Mbps**
15:20:35.753	3G-SM	PDP context deactivation request: QoS unsupported
15:20:35.754	3G-SM	PDP context deactivation accept
15:20:42.123	3G-SM	PDP context setup request
15:20:43.278	3G-SM	PDP context setup accept: dlink peak tput=**2Mbps**
...	3G-SM	...

Table 13: A network failure due to unsupported prepaid QoS configuration (maximum downlink peak throughput throttled to 128Kbps).

The second observed instance is a prepaid data plan violation in AT&T. According to AT&T's contract [51], when the prepaid user runs out of its high-speed data, (s)he can still retain the low-speed data connectivity (throttled to 128Kbps). But `NetDiag` reports that, this policy can be violated if the prepaid user is associated with a 3G private Femtocell; *i.e.*, (s)he cannot gain low-speed data connectivity after running out of its high-speed data. Table 13 shows the traceback logs in this scenario. This failure is caused by the network-side misconfiguration for prepaid user QoS throttling. The phone in 3G Femtocell first attempts to activate its data session. The network accepts this request, and guarantees to provide 2Mbps downlink peak throughput. However, this guarantee exceeds the maximum speed (128Kbps) when running out of high-speed data. Upon detecting this violation, the core network deactivates the data session immediately. But the next time the device requests for data session activation, the core network still assigns unsupported downlink peak throughput to it. The device thus keeps on receiving deactivation session requests from the network, and is stuck at the inactivity state of the 3G-SM protocol. With this hint, the mobile device can report these issues to network operators, and help them resolve the network failures at fine-granularity.

Security loophole detection. This case leverages MOBILEINSIGHT to detect security loopholes over cellular networks. We built a security checker with 3G/4G MM analyzers of `LteNasAnalyzer` and `UmtsNasAnalyzer`. It aims to detect if the signaling/data communication between mobile device and the cellular base stations are encrypted and integrity protected. To this end, it tracks the authentication and key agreement (AKA) procedure [10], and the security mode activation in 3G/4G. Both procedures are carried over the 3G/4G mobility management (MM/EMM) layer. It checks the authentication status in the registration process, and the ciphering/integrity protection algorithms specified in the security mode commands. If the authentication fails or the ciphering/integrity algorithms are not activated, the user's

Category	Examples	In-Device	COTS	Coverage	Granularity	Analysis	Runtime	API
Our approach: MOBILEINSIGHT		√	√	**Almost full**	**Fine-grained (msg-level)**	√	√ (ms-level)	√
OS API	[6, 7, 52–54]	√	√	Limited (mainly service states)	coarse (aggregated)	×	√	√
RIL Analyzer	[55]	√	√	Limited (depending on AT cmds)	coarse	√ (partially)	√	×
PC-side Debugger	[2–4, 23]	× (PC)	√	Full	fine (msg-level)	×	√	×
Radio analytics	[5, 56]	× (USRP)	×	Limited (partial PHY)	Selected PHY fields	√ (partial PHY)	√	N/A
RRC inference	*e.g.*, [31–33, 55]	√	√	Limited	Selected (RRC states)	√ (RRC only)	√ (mainly offline)	N/A
SnoopSnitch	[57]	√	√	Limited (IMSI catcher detection)	Selected	√ (Security only)	√	×

Table 14: Comparison of device-side solutions to retrieve and infer cellular network information.

Timestamp	Protocol	Event
02:21:24.064	4G-EMM	Attach req: support {EEA0,EEA2}
02:21:24.469	4G-EMM	Security mode command: use **EEA0**
02:21:24.470	4G-EMM	Security mode complete
02:21:24.519	4G-EMM	Attach accept

Table 15: Null signaling encryption loophole in China Mobile.

Timestamp	Protocol	Event
00:45:13.818	3G-RRC	Meas control: monitor 3G and 2G
00:45:14.140	3G-RRC	Meas report: 2G ARFCN=401, RSSI=-80dB
17:57:15.130	3G-RRC	**Handoff command**: to 2G ARFCN=401
17:57:15.410	3G-SM	Meas report: 3G Freq=4360, RSCP=-90dBm

Table 16: Event log of an FCFS handoff strategy in a AT&T's 3G cell.

signaling or data may not be well protected over the air. In this case, a vulnerability would be reported.

We have applied our tool and found a new security loophole in China Mobile on its 4G mobility management protocol (EMM). As shown in Table 15, after registration (attach) to the network, EMM configures the device to not encrypt the signaling messages (EEA0 algorithm). This results in the man-in-the-middle sniffing on user behaviors and privacy intrusion. Our tool provides warnings to the device. We are contacting the operator to fix this vulnerability.

Performance boost. The next showcase is a handoff advisor that advises whether a better-performing cell is available for handoffs. It is built with `LteRrcAnalyzer` and `WcdmaRrcAnalyzer`, which implement our handoff decision logic inference algorithms (§5.2). By inferring the protocol operation logic, this tool helps to forecast the potential sub-optimal handoff behaviors from the network and reconfigure the device to work them around. To this end, it leverages the current control state and measurement reports. If the measurement reports indicate that a faster cell (*e.g.* 4G) exists, but the predicted target cell for handoff is slower (*e.g.* 3G), it alerts the device with a suboptimal handoff.

We show a real instance on how the tool helps the device to prevent suboptimal handoffs. The instance occurs when certain AT&T 3G cell makes a handoff decision too early so that it migrates the device to a low-performance cell (2G here). Table 16 shows this scenario. The device is initially served by a 3G cell. To initiate a handoff, the 3G cell asks the device to measure both 2G and 3G cells. The problem occurs when both 2G and 3G cells have good signal strengths. Moreover, the 3G cell uses the first-come-first-serve (FCFS) handoff strategy. Consequently, the serving cell may immediately hand over the device to 2G upon receiving the 2G report first, without waiting for the 3G measurement report. It thus misses the desired handoff to 3G. Our tool captures this issue by inferring this 3G cell's decision logic, and reports the suboptimal handoff *before* it is triggered. Given this advice, we eliminate this suboptimal handoff by disabling 2G (via secret codes) at the device.

9 Related Work

We first compare MOBILEINSIGHT with other approaches to learning cellular information from the device. Table 14 summarizes the features and limitations of each scheme. It shows that MOBILEINSIGHT is the only software-only in-device cellular network analyzer, which covers more 3G/4G control-plane and low-layer protocols, supports both fine-grained information collection and protocol analytics at runtime, operates on COTS phones , *and* offers APIs for mobile applications. There are also in-device analyzers [58–60], but they focus on application and transport layers, not cellular-specific lower-layers.

Meanwhile, extensive research has been conducted to improve device-side performance over cellular networks, including video adaptation [56], energy saving [61–63], and cellular congestion control [64, 65], *etc.* They could also benefit from MOBILEINSIGHT with further access to the fine-grained, runtime information. Finally, there are ongoing efforts on optimizations for handoff [37–40], software-defined LTE [66–70] and backend cellular infrastructure [71–73]. The insights from MOBILEINSIGHT over operational networks (*e.g.*, signaling protocols and handoff policies), can help to better design the future network infrastructure.

10 Conclusion

The cellular network provides more control utilities than the wired Internet, including radio resource control, security, mobility support, and carrier-grade services, to name a few. Understanding these functions and their protocol operations will be important for refining the design and optimizing application performance. However, such fine-grained protocol operations have remained inaccessible to the research community.

MOBILEINSIGHT represents our first effort to build a software tool to open up the blackbox operations. It enables open access to the low-level protocol operations in 3G/4G from the device side. It runs on the COTS phone, but leverages its increasing capability. It directly extracts the signaling and/or low-layer messages from the side channel toward 3G/4G hardware interface, decodes the protocol messages, and infers the protocol state dynamics and decision logic at runtime through analyzers. Through MOBILEINSIGHT's APIs, applications can benefit from accessing such low-level domain knowledge. In presence of network failures, security loopholes, or performance degrade, MOBILEINSIGHT helps to detect the problematic instances, infer the root causes, and suggest fixes.

In the broader context, MOBILEINSIGHT is designated to be an open, extensible tool *for the community* and *by the community*. It may help us to examine cellular networks in the large-scale setting via crowdsourcing. More community efforts are clearly needed to enhance and extend every aspect, particularly analyzers and applications atop. The collected datasets can further be shared within the community. Our own experience so far has confirmed that such tool-building efforts are quite worthwhile and can be rewarding.

Acknowledgments: We thank the anonymous reviewers and shepherd for their constructive comments. This work is supported in part by NSF awards (CNS-1526456, CNS-1526985, CNS-1423576 and CNS-1421440).

11 References

[1] Cisco Visual Networking Index. Global Mobile Data Traffic Forecast Update, 2014–2019, 2015.

[2] Qualcomm. QxDM Professional - QUALCOMM eXtensible Diagnostic Monitor. http://www.qualcomm.com/media/documents/tags/qxdm.

[3] Xcal-mobile. http://www.accuver.com.

[4] MTK Catcher. http://www.finetopix.com/showthread.php/40844-MTK-Catcher.

[5] S. Kumar, E. Hamed, D. Katabi, and L. Erran Li. LTE Radio Analytics Made Easy and Accessible. In *ACM SIGCOMM*, 2014.

[6] Android.telephony. http://developer.android.com/reference/android/telephony/package-summary.html.

[7] Android telephonymanager class. http://developer.android.com/reference/android/telephony/TelephonyManager.html.

[8] Google. Project fi, 2015. https://fi.google.com/about/.

[9] 3GPP. TS24.008: Mobile Radio Interface Layer 3, 2012.

[10] 3GPP. TS24.301: Non-Access-Stratum (NAS) for EPS; , Jun. 2013.

[11] 3GPP. TS25.331: Radio Resource Control (RRC), 2006.

[12] 3GPP. TS36.331: Radio Resource Control (RRC), 2012.

[13] 3GPP. TS36.211: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical channels and modulation.

[14] 3GPP. TS36.212: Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding.

[15] 3GPP. TS36.213: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.

[16] 3GPP. TS36.321: Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification, Mar. 2014.

[17] 3GPP. TS36.322: Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification, Sep. 2012.

[18] 3GPP. TS36.321: Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification, Jun. 2014.

[19] 3GPP. TS27.007: AT command set for User Equipment (UE), 2011.

[20] Nexus 5 field test mode. https://play.google.com/store/apps/details?id=com.cellmapper.nexus5fieldtestmode&hl=en.

[21] Field test mode: What it is and how to enable it on your phone. http://www.ubersignal.com/field-test-mode.

[22] Android platform development kit: Radio layer interface. http://www.netmite.com/android/mydroid/development/pdk/docs/telephony.html.

[23] xgoldmon. https://github.com/2b-as/xgoldmon.

[24] Android source code for usb tethering. https://android.googlesource.com/kernel/msm.git/+/android-6.0.0_r0.9/drivers/usb/gadget/android.c.

[25] Android source code for qualcomm cellular diagnostic mode. https://android.googlesource.com/kernel/msm.git/+/android-6.0.0_r0.9/drivers/char/diag/.

[26] Android source code for meadiatek cellular diagnostic mode. https://android.googlesource.com/kernel/mediatek/+/android-4.4.4_r3/drivers/misc/mediatek/.

[27] ios baseband commands. https://www.theiphonewiki.com/wiki/Talk:Baseband_Commands.

[28] 3GPP. TS36.300: E-UTRA and E-UTRAN; Overall description; Stage 2, 2011.

[29] Wikipedia: Abstract syntax notation one (asn.1). https://en.wikipedia.org/wiki/Abstract_Syntax_Notation_One.

[30] K. Sandlund, G. Pelletier, and L. Jonsson. The robust header compression (rohc) framework, 2010. RFC 5795.

[31] S. Rosen, H. Luo, Q. A. Chen, Z. M. Mao, J. Hui, A. Drake, and K. Lau. Discovering fine-grained rrc state dynamics and performance impacts in cellular networks. In *MobiCom*, 2014.

[32] J. Huang, F. Qian, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *ACM MobiSys*, 2012.

[33] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Characterizing Radio Resource Allocation for 3G Networks. In *IMC*, 2010.

[34] ZTE UMTS Handover Description. http://www.slideshare.net/quyetnguyenhong/zte-umtshandoverdescription.

[35] Netmanias. Overview of LTE handover. http://www.netmanias.com/en/post/techdocs/6224/emm-procedure-6-handover-without-tau-part-1-overview-of-lte-handover.

[36] Blind handover. https://www.linkedin.com/groups/Can-anybody-explain-what-exactly-1180727.S.158571676.

[37] K. Dimou, M. Wang, Y. Yang, M. Kazmi, A. Larmo, J. Pettersson, W. Muller, and Y. Timner. Handover within 3gpp lte: design principles and performance. In *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th*, pages 1–5. IEEE, 2009.

[38] T. Jansen, I. Balan, J. Turk, I. Moerman, and T. Kurner. Handover parameter optimization in lte self-organizing networks. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, pages 1–5. IEEE, 2010.

[39] A. Lobinger, S. Stefanski, T. Jansen, and I. Balan. Coordinating handover parameter optimization and load balancing in lte self-optimizing networks. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1–5. IEEE, 2011.

[40] L. Korowajczuk. *LTE, WiMAX and WLAN network design, optimization and performance analysis*. John Wiley & Sons, 2011.

[41] P. Dupont, B. Lambeau, C. Damas, and A. v. Lamsweerde. The qsm algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22(1-2):77–115, 2008.

[42] N. Walkinshaw, K. Bogdanov, M. Holcombe, and S. Salahuddin. Reverse engineering state machines by interactive grammar inference. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, pages 209–218. IEEE, 2007.

[43] C. Damas, B. Lambeau, P. Dupont, and A. Van Lamsweerde. Generating annotated behavior models from end-user scenarios. *Software Engineering, IEEE Transactions on*, 31(12):1056–1073, 2005.

[44] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. 1992.

[45] Mobileinsight. http://metro.cs.ucla.edu/mobile_insight.

[46] Python-for-android project. https://python-for-android.readthedocs.org/en/latest/.

[47] AndroidHeadlines. Over 27.44% users root their phone(s) in order to remove built-in apps. http://www.androidheadlines.com/2014/11/50-users-root-

phones-order-remove-built-apps-one.html.

[48] 3GPP. TS36.508: Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Packet Core (EPC); Common test environments for User Equipment (UE) conformance testing, Dec 2015.

[49] Cross-validation (statistics). https://en.wikipedia.org/wiki/Cross-validation_(statistics).

[50] Monsoon power meter. https://www.msoon.com/LabEquipment/PowerMonitor/.

[51] At&t's prepaid data plan policy. https://www.att.com/shop/wireless/plans/voice/sku7420265.html.

[52] ios developer library: Core telephony framework reference. https://developer.apple.com/library/prerelease/ios/documentation/NetworkingInternet/Reference/CoreTelephonyFrameworkReference/index.html.

[53] Connection manager. https://msdn.microsoft.com/en-us/library/bb416435.aspx.

[54] Windows phone: Telephony api. https://msdn.microsoft.com/en-us/library/aa922068.aspx.

[55] N. Vallina-Rodriguez, A. Auçinas, M. Almeida, Y. Grunenberger, K. Papagiannaki, and J. Crowcroft. Rilanalyzer: a comprehensive 3g monitor on your phone. In *IMC*, 2013.

[56] X. Xie, X. Zhang, S. Kumar, and L. E. Li. pistream: Physical layer informed adaptive video streaming over lte. In *MobiCom*, 2015.

[57] Snoopsnitch. https://opensource.srlabs.de/projects/snoopsnitch.

[58] N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, N. Weaver, and V. Paxson. Beyond the radio: Illuminating the higher layers of mobile networks. In *Mobisys'15*, pages 375–387. ACM, 2015.

[59] A. Nikravesh, H. Yao, S. Xu, D. Choffnes, and Z. M. Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *Mobisys'15*, pages 389–404. ACM, 2015.

[60] C. Shepard, A. Rahmati, C. Tossell, L. Zhong, and P. Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[61] C. Shi, K. Joshi, R. K. Panta, M. H. Ammar, and E. W. Zegura. Coast: collaborative application-aware scheduling of last-mile cellular traffic. In *Mobisys'14*, pages 245–258. ACM, 2014.

[62] X. Chen, A. Jindal, N. Ding, Y. C. Hu, M. Gupta, and R. Vannithamby. Smartphone background activities in the wild: Origin, energy drain, and optimization. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 40–52. ACM, 2015.

[63] D. H. Bui, Y. Liu, H. Kim, I. Shin, and F. Zhao. Rethinking energy-performance trade-off in mobile web page loading. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pages 14–26. ACM, 2015.

[64] F. Lu, H. Du, A. Jain, G. M. Voelker, A. C. Snoeren, and A. Terzis. Cqic: Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 45–50. ACM, 2015.

[65] K. Winstein, A. Sivaraman, H. Balakrishnan, et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *NSDI*, pages 459–471, 2013.

[66] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[67] Openlte implementation. http://sourceforge.net/projects/openlte/.

[68] L. E. Li, Z. M. Mao, and J. Rexford. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 7–12, 2012.

[69] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: scalable and flexible cellular core network architecture. In *CoNEXT*, 2013.

[70] M. Arslan, K. Sundaresan, and S. Rangarajan. Software-defined networking in cellular radio access networks: potential and challenges. *Communications Magazine, IEEE*, 53(1):150–156, 2015.

[71] A. Iyer, L. E. Li, and I. Stoica. Celliq: real-time cellular network analytics at scale. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 309–322, 2015.

[72] A. Banerjee, J. Cho, E. Eide, J. Duerig, B. Nguyen, R. Ricci, J. Van der Merwe, K. Webb, and G. Wong. Phantomnet: Research infrastructure for mobile networking, cloud computing and software-defined networking. *GetMobile: Mobile Computing and Communications*, 19(2):28–33, 2015.

[73] Z. Li, W. Wang, T. Xu, X. Zhong, X.-Y. Li, Y. Liu, C. Wilson, and B. Y. Zhao. Exploring cross-application cellular traffic optimization with baidu trafficguard. 2016.