UCLA CS211 17F Project Report

Project 2: Mobile Analytics with Intel LTE Baseband Chipset

Kiran Sivakumar*
UCLA
ksivakumar@cs.ucla.edu

Siva Kesava Reddy K
UCLA
sivakesava@cs.ucla.edu

Andrew Chen
UCLA
andrewc@cs.ucla.edu

Vishrant Vasavada
UCLA
v.vasavada@hotmail.com

ABSTRACT

MobileInsight[1] supports runtime monitoring of cellular protocol information for rooted Android phones based on Qualcomm and MediaTek 3G/4G chipsets. It is an in-device software tool that doesn't require any external hardware or additional support. In this report, we show our attempts to generalize and extend MobileInsight to support Intel's LTE baseband chipset. Intel's modem log files use the ISTP format. We discuss here the various existing tools and approaches we tried to understand these ISTP formatted logs so that we could decode them to make them more human-readable. While we were unsuccessful in building a parser for ISTP format, this report will still help anyone who works on the same project in future to eliminate trying out various possibilities that we have already tried and save their time or to get some hints from our work.

1 INTRODUCTION

Usually, the cell phone apps and OS provide limited functionality to access low-level cellular protocols logs at a runtime. This is a problem because without this information it is impossible for end users to understand how the network operates. Because of this, end users are unable to tell if the network is operating appropriately and efficiently and are unable to suggest alternative cellular protocols. The network remains like a black box to users.

Many researchers think that an open access to fine-grained runtime network information will allow them to assess various real-life instances such as why device experienced a hand-off, whether it was a good decision, and other such information. For example, iCellular[2] uses such low-level logs to develop a multi-carrier access infrastructure to allow high-quality cellular network service anytime and anywhere to its users. iCellular shows us how low-level logging enabled them to achieve active monitoring and intelligent selection (hand-off) when compared to passive monitoring and unwise selection (hand-off) in Google Fi.

MobileInsight is a software tool that enables runtime monitoring and analytics of such cellular protocol logs without additional hardware. It already has a support to extract and parse raw binary logs, extracted through the diagnostic mode of Qualcomm and MediaTek's chipsets. However, it is unable to extract and parse for Intel's LTE baseband chipset and our project aims to solve this limitation. Since Intel, Qualcomm and MediaTek are only the three major players in cellular chipset world, solving this limitation would allow MobileInsight to run on pretty much any commercial-off-the-shelf (COTS) device.

Typically, each cellular network message contains metadata header and payload. These metadata header formats are specific to each chipset. Therefore, we cannot use the existing MobileInsight's parsers to parse binary logs for Intel's LTE baseband chipsets as it was developed only for Qualcomm and MediaTek chipsets.

Another major problem with the messages is that they have padding bits which makes it difficult to understand when a new message starts. There will also be some chip-based debugging messages which are interleaved with the actual messages in the cellular logs that we obtain which are irrelevant to us. Separation of such chip based messages is also a difficult task when the logs are in binary format. Additionally, we haven't come across any open-source documentation for Intel's cellular protocol messages format. Without proper tools and documentation, it is increasingly difficult to infer metadata headers and extract message type and other information. Our task here is to identify and decode Intel's messages so that we can perform analysis on network behavior.

We started with a goal to develop a MobileInsight monitor and message parser plugins for the Intel LTE baseband chipset. The MobileInsight monitor would extract the raw binary logs from the Intel device's diagnostic port and feed them to the message parser. The parser would then parse and analyze the raw logs to resolve the 3G/4G over-the-air messages from the baseband. The monitor and parser will only target data associated with the top three MobileInsight defined network layers: Session Management, Mobility Management, and Radio Resource Control. Typically, Physical (L1) and Link (L2) layer message headers in wireless networks are compressed and they may need parameters from Radio Resource Control to decompress and thus decode. Hence, unless we successfully parse Radio Resource Control messages, we won't be able to decode L1 and L2 messages and so we limit our goal to only top three layers for this project. Decoding L1 and L2 messages could then be done in future work.

However, as you will further see in this report, we were unsuccessful in meeting our this initial goal. Intel modem logs are in ISTP (Intel) format. The ISTP log file contains hex dump which none of the generic text editors could decode and display fully in UTF-8 format. We tried several approaches to the understanding of ISTP format but could not crack it. In the end, we limited the scope of our project to just creating the MobileInsight monitor for ASUS's Intel chipset phones and manually read logs to check for any patterns.

*Group 6

This report describes all the tools and approaches that we used to try and decode ISTP logs and the results we got. We hope that this work will help researchers working on the same project to narrow down their approaches or get some hints from our results.

The further report is divided into 6 sections. In section 2, we talk about the related prior work done or tools developed to extract and parse modem logs. We also briefly mention their limitations. Section 3 gives brief background and specifications of the testing device that we used throughout this project. Section 4 talks about our approaches to solving this problem. It includes information about the scripts that we used to extract logs, monitor plugin we developed for MobileInsight, results or conclusions of the manual analysis we did of the ISTP logs, existing tools that we used to try and parse ISTP logs and further efforts and approaches towards the work. Section 4 describes the future work that could be done towards our initial goal. Finally, we conclude our report in section 5 and give references in section 6. We also include an appendix in the end to address the questions we were asked during the final project presentation.

2 RELATED WORKS

Many schemes have been proposed previously to learning cellular information from the device. However, most of these schemes, including MobileInsight, covers only 3G/4G protocols.

xgold-mon[3] project did attempt to work on parsing the Intel LTE baseband chipset logs to convert them to GSM/UMTS radio messages sent over the air so that they could be monitored using a software such as *wireshark*. However, *xgold-mon* wasn't able to offer protocol analytics. Also, it worked only with PCs and lacked in-device collection technique.

Inspired by *xgold-mon* project, *xgoldscanner*[4] was developed. But it was only limited to the collection of 2G and 3G traces from the phones.

GSMmap[5] is another android application which supports in-device data collections. This GSM Security Map application compares the protection capabilities of mobile networks for which it collects 2G and 3G network traces. This collected data is then submitted to gsmmap.org for the analysis.

Both the projects mentioned above only support old Samsung devices such as Galaxy S2/S3 and Galaxy Note 2. This is because Samsung's Service Mode Application API can provide more generous access to baseband information than the standard Android API. Hence, much of the logs obtained through USB logging mode could easily be compared with the logs obtained by using the API.

Other similar examples to *xgold-mon* are QXDM for Qualcomm baseband and MTK Catcher for MediaTek baseband.

3 TESTING DEVICE

We were given Asus Zenfone 2E US-AT&T Exclusive android phone to work with. This phone uses Intel Atom Z2560 processor and Intel 7160 modem[6]. Intel XMM 7160 modem comes with LTE[7] which is exactly what we need. This rooted android device also came with MobileInsight app and *modemtrace* script. We will be discussing *modemtrace* script in our next section. The device also had inactive AT&T SIM card. Hence, we weren't able to get any data transfer initially in the phone.

However, our project required us to compare logs collected using both active SIM and inactive SIM as we'll see later in the report. Since we weren't provided with active AT&T SIM card, we tried to put in our active T-Mobile SIM. The phone still wasn't able to recognize any SIM. It was AT&T phone but we believe that it should have at least detected a SIM which it didn't. In fact, doing this actually crashed our test device. The modem communication ports (/dev/gsmtty*) vanished. Inserting a working AT&T SIM didn't fix the issue. We also noticed that baseband version, IMEI number and other such informations were all set to "unknown". This is when we realized that the test phone we were given actually had some issues in OS/Kernal.

The next step was to fix the phone so that our project doesn't come at a halt. For this, we needed to re-install OS. Our phone had AT&T's ZE500CL SKU firmware. On ASUS website, we did find firmware images for ZE500CL. However, this was for WW version and flashing it using TWRP Recovery didn't work. It gave us the error - *This package is for ASUS_Z00D; it is Z00D*. We figured out that we'll have to change the recovery image if we wanted to make this firmware zip to work. Hence, we obtained ASUS Z00D WW Recovery Image and changed our phone's recovery image to it. This changed our boot, recovery and droidboot images to WW SKU Version. We then flashed the firmware we used before and it fixed the phone entirely. We were able to Baseband version, IMEI and other other information. We were also able to see /dev/gsmtty* modem communication ports. The SIM card was also being recognized and we could get the data transfer working with the active sim. This way, as a part of our project, we were also able to fix the test device we were initially provided.

4 RESEARCH AND RESULTS

In this section, we describe the various approaches and techniques we tried to extract and parse the cellular network logs of Intel's LTE baseband chipset in our test device.

We divide this section into 4 sub-sections. Section 4.1 describes our log extraction methodology and results. We also describe our implementation of the MobileInsight monitor for Asus Intel baseband phones. The next section (4.2) talks about various existing tools that we tried on our extracted logs to parse ISTP format to human-readable format. In section 4.3, we give introduction to and talk about AT Commands. Finally, we wrap up our research in section 4.4 by giving analysis of the ISTP logs.

4.1 Log Extraction

There were some prerequisites involved to the log extraction process. We list the two prerequisites below:

- Install ADB Drivers on computer if not done already
- Turn on Developers Mode on the phone and turn on USB Debugging

To extract logs, we then attached our test device to our computer with ADB drivers installed. The following three commands were then executed:

(1) adb shell
(2) su
(3) ./modemtrace-starter.sh

The debug messages of *modemtrace-starter* script showed us the output directory for ISTP log files. It was easy to retrieve the log files from here to our computer for analysis.

We now describe *modemtrace* script in more details. *modemtrace* script essentially sets all the required properties for logging. For e.g., it sets the output directory for ISTP log files. It also sets modem logging levels like debug, verbose, etc. After doing all these required configurations, it calls *asus_mts* script. *asus_mts* script is then responsible to extract and save all the modem ISTP logs in the directory specified by *modemtrace* script. This *asus_mts* script is the one where all the magic happens. To know what *asus_mts* actually does, we tried to search for it and found its compiled binary version on github[8]. However, we could not find its source code anywhere. We also tried to reverse engineer this binary into C++ code using *IDA - The Interactive Disassembler v7.0 Demo* software. Unfortunately, the free demo version did not give us C++ code but only showed translated assembly code. It was hard for us to follow through this code and make anything out of it as none of us had enough knowledge on assembly language.

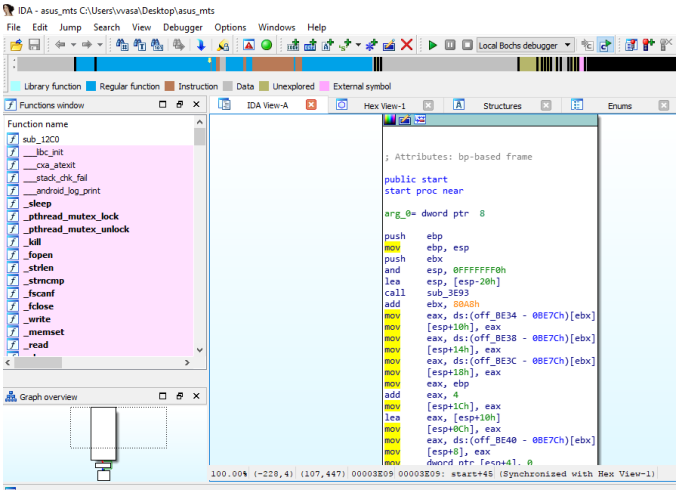

Figure 1: Assembly code output of *asus_mts* binary

5 TIMELINE

The project can be decomposed into three steps: setup message extraction on the mobile device, identify which binaries are relevant *RRC*, *SM*, *MM* and their formats, and the actual implementation of the previous step. A timeline is provided in table 1.

1. Setup message extraction	Week 3
2. Identify message formats	Weeks 4,5, half of 6
3. Implementing monitor and parser	Weeks 6,7,8
4. Final report and presentation write-up	Weeks 9, 10

Table 1: Project Timeline

REFERENCES

[1] Li, Yuanjie, et al. "Mobileinsight: extracting and analyzing cellular network information on smartphones." MobiCom. 2016.

[2] Li, Yuanjie, et al. "iCellular: Device-Customized Cellular Network Access on Commodity Smartphones." Usenix. 2016.

[3] Engel, Tobias. fiXgoldmon.fi GitHub, 30 Dec. 2013, github.com/2b-as/xgoldmon.

[4] SRLabs Open Source Projects. "xgoldscanner." Mobile Network Assessment Tools, opensource.srlabs.de/projects/mobile-network-assessment-tools/wiki/Xgoldscanner.

[5] SRLabs Open Source Projects. "GSMmaps-apk." Mobile Network Assessment Tools, opensource.srlabs.de/projects/mobile-network-assessment-tools/wiki/GSMmap-apk.

[6] Asus Zenfone 2E. https://www.asus.com/Phone/ZenFone_2_ZE500CL/specifications/

[7] Intel XMM 7160. https://ark.intel.com/products/66646/Intel-XMM-7160-Slim-Modem

[8] ASUS MTS Script. https://github.com/artefvck/android_device_asus_a400cg/blob/master/logtoold/asus