MobileInsight-Cloud

(Cloud service and Web interface of MobileInsight)

Yi-Chun Lo, Lily Lin, Chun-Yao Chang, Po-Nien Kung

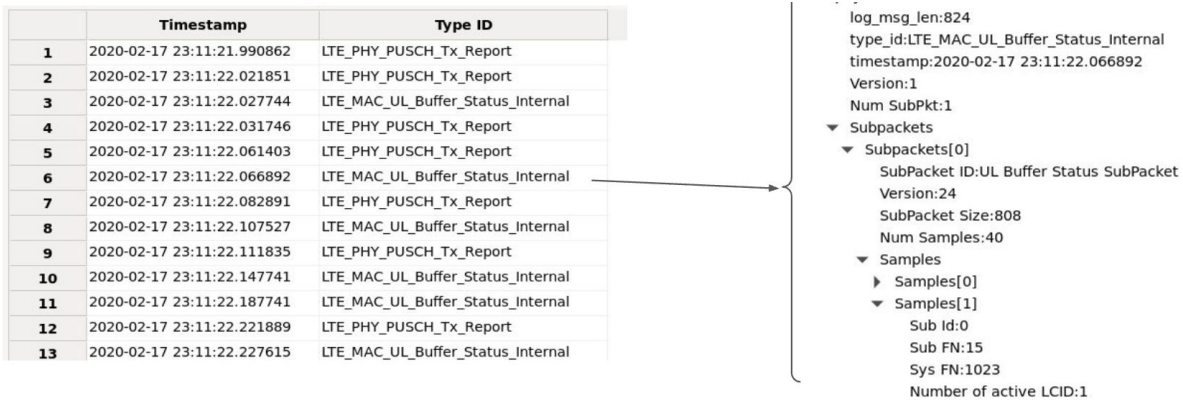
Outline

1. Overview
2. Data Transform
3. Data Filters
4. MobileInsight-Cloud
5. Demo
6. Limitations and Future Work

Overview

MobileInsight

- A software tool which enables runtime cellular network monitoring and analytics on smartphones.
- Providing open access (in software) to fine-grained cellular information on 5G protocols.
- Facilitate researchers and developers to readily and quickly obtain the low-level network information through easy-to-use APIs.



	Timestamp	Type ID
1	2020-02-17 23:11:21.990862	LTE_PHY_PUSCH_Tx_Report
2	2020-02-17 23:11:22.021851	LTE_PHY_PUSCH_Tx_Report
3	2020-02-17 23:11:22.027744	LTE_MAC_UL_Buffer_Status_Internal
4	2020-02-17 23:11:22.031746	LTE_PHY_PUSCH_Tx_Report
5	2020-02-17 23:11:22.061403	LTE_PHY_PUSCH_Tx_Report
6	2020-02-17 23:11:22.066892	LTE_MAC_UL_Buffer_Status_Internal
7	2020-02-17 23:11:22.082891	LTE_PHY_PUSCH_Tx_Report
8	2020-02-17 23:11:22.107527	LTE_MAC_UL_Buffer_Status_Internal
9	2020-02-17 23:11:22.111835	LTE_PHY_PUSCH_Tx_Report
10	2020-02-17 23:11:22.147741	LTE_MAC_UL_Buffer_Status_Internal
11	2020-02-17 23:11:22.187741	LTE_MAC_UL_Buffer_Status_Internal
12	2020-02-17 23:11:22.221889	LTE_PHY_PUSCH_Tx_Report
13	2020-02-17 23:11:22.227615	LTE_MAC_UL_Buffer_Status_Internal

Limitation

- MobileInsight only works in local machine.
  - Can’t share log with other users.
- Need to deal with complicated dependencies when installing.
  - Especially for Windows and Mac with M-series chip.
- Lack of features in current GUI.
  - No filters
  - No download option

Solution

- Provide a cloud version of MobileInsight - **MobileInsight Cloud**
  - Can filter, upload, and download log.
  - No need for installing MobileInsight.
- Use Redis as log-stored database.
  - In-memory database, **fast**
  - Key-value storage, **high scalability**

Data Transform

Data Transform

- Convert the raw mi2log log file to JSON object and store in Redis.
  - JSON object is easy to display.
- Key: `f'{filename}:{type_id}:{timestamp}:{order}'`
  - Get enough information of the file without loading all the data.
  - Allows us to filter more efficiently without accessing the content
- Value: log file with json format
  - Extend the Analyzer class in mobile_insight library to process mi2log format.
  - Parse the raw XML string to XML object in “Msg” field.
  - Add “order” field to preserve the order of the log item.
- Also store the raw mi2log file
  - For user to download the mi2log file.


```
{
  "log_msg_len" : 46
  "type_id" : "LTE_RRC_OTA_Packet"
  "timestamp" : "2020-11-16 09:51:21.245180"
  "Pkt Version" : 26
  "RRC Release Number" : 15
  "RRC Version Number" : "0x50"
  "NR RRC Release Number" : 15
  "NR RRC Version Number" : "0x60"
  "Radio Bearer ID" : 0
  "Physical Cell ID" : 132
  "Freq" : 38950
  "SysFrameNum/SubFrameNum" : 2351955968
  "PDU Number" : 7
  "SIB Mask in SI" : 0
  "Msg Length" : 13
  "Msg" : {...}
  "order" : 442
} +
```

Data Filters

Data Filter

- Originally, MobileInsight **only** support time-based and type_id filtering:
  - Time: python time.time()
  - Type_id
    - `LTE_RRC_OTA_Packet`
    - `LTE_NAS_ESM_OTA_Incoming_Packet`
- Requirement: We need to add **timestamp** as a new filtering feature.
  - `start_timestamp = datetime(2020, 11, 16, 9, 49, 0)`
  - `end_timestamp = datetime(2020, 11, 16, 9, 50, 0)`
  - `Need to modify MobileInsight APIs to process timestamp`

New Feature: Timestamp

- We add the timestamp as new feature by tracing through and modifying the following code: offline_replayer.py

```
def enable_log(self, type_name):
    """
    Enable the messages to be monito

    If this method is never called,
```

```
def enable_log(self, type_name, start_timestamp = None, end_timestamp = None):
    """
    Enable the messages to be monitored. Refer to cls.SUPPORTED_TYPES for suppo

    If this method is never called, the config file existing on the SD card wil
```

Here the timestamp is a python datetime object.

Looks simple, but, needs several steps to work.

Continue Tracing

- dm_collector_c.cpp
  - `dm_collector_c_set_filtered`
  - `dm_collector_c_receive_log_packet`
- export_manager.cpp
  - `manager_export_binary`
  - `manager_set_filter`

Add time interval attribute in **export manager**

- export_manager.h
  - `struct ExportManagerState`

```
bool
manager_export_binary (struct ExportManagerState *pstate, const char *b, si

    double packet_timestamp = 0.0;
    int type_id = get_log_type_and_timestamp(b, length, packet_timestamp);

    if (pstate->whitelist.count(type_id) > 0 &&
        packet_timestamp >= pstate->start_timestamp &&
        packet_timestamp <= pstate->end_timestamp) { // filter
```

```
// Manage the output of logs.
struct ExportManagerState {
    FILE *log_fp;   // Point to
    std::string filename;
    std::set<int> whitelist;
    double start_timestamp;
    double end_timestamp;
};
```

Compute Timestamp

- Extract timestamp from packet and convert it to UNIX format

```
uint64_t
get_timestamp(const char *b, size_t offset) {
    // 8 byte offset
    uint64_t timestamp = 0;
    std::memcpy(&timestamp, b + offset, sizeof(uint64_t));
    return timestamp;
}

double
convert_to_unix(uint64_t packet_timestamp) {
    int seconds = int(double(packet_timestamp) / PER_SECOND);
    int useconds = (double(packet_timestamp) / PER_USECOND) - double(seconds) * 1.0e6;
    return BASE_TIMESTAMP + seconds + useconds / 1.0e6;
}
```

```
case QCDM_TIMESTAMP: {
    const double PER_SECOND = 52428800.0;
    const double PER_USECOND = 52428800.0 / 1.0e6;
    assert(fmt[i].len == 8);
    // Convert to a Python long integer object
    // unsigned long long iiii = *((unsigned long long *) p);
    unsigned long long iiii = 0;    //Yuanjie: FIX crash on Android
    int seconds = int(double(iiii) / PER_SECOND);
    int useconds = (double(iiii) / PER_USECOND) - double(seconds) * 1.0e6;
    PyObject *epoch = PyDateTime_FromDateAndTime(1980, 1, 6, 0, 0, 0, 0);
    PyObject *delta = PyDelta_FromDSU(0, seconds, useconds);
    decoded = PyNumber_Add(epoch, delta);
    n_consumed += fmt[i].len;
    Py_DECREF(epoch);
    Py_DECREF(delta);
    break;
}
```

- log_packet.cpp
- export_manager.cpp

MobileInsight-Cloud

MobileInsight-Cloud

- Powered by Streamlit - a Python library for web applications.
  - All in pure Python. No front-end experience required.
  - Easy deployment
  - Strong open-source community

MobileInsight-Cloud

Display Upload

Filename

20201116_175917_Xiaomi-Mi10_46000.mi2log

type_id

LTE_NAS_ESM_S... × LTE_NAS_EMM_... × LTE_RRC_OTA_P... ×

timestamp

2020-11-16 09:48:42 → 2020-11-16 09:59:18

Download Filtered mi2log file | Download Filtered JSON | Download Selected JSON

		filename	type_id	timestamp	or
☑	0	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:42	
	1	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:42	
	2	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:43	
	3	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:43	
	4	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:44	
	5	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:44	
	6	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:45	
	7	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:45	
	8	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:45	
	9	20201116_175917_Xiaomi-Mi10_46000.mi2log	LTE_RRC_OTA_Packet	2020-11-16 09:48:45	


```
{
  "log_msg_len" : 46
  "type_id" : "LTE_RRC_OTA_Packet"
  "timestamp" : "2020-11-16 09:48:42.525084"
  "Pkt Version" : 26
  "RRC Release Number" : 15
  "RRC Version Number" : "0x50"
  "NR RRC Release Number" : 15
  "NR RRC Version Number" : "0x60"
  "Radio Bearer ID" : 0
  "Physical Cell ID" : 132
  "Freq" : 39148
  "SysFrameNum/SubFrameNum" : 2888826880
  "PDU Number" : 7
```

Demo

Limitation and Future Work

Timestamp

- Recall that we compute timestamp with the following conversion:

```
const double PER_SECOND = 52428800.0;
const double PER_USECOND = 52428800.0 / 1.0e6;
const time_t BASE_TIMESTAMP = 315993600;

uint64_t
get_timestamp(const char *b, size_t offset) {
    // 8 byte offset
    uint64_t timestamp = 0;
    std::memcpy(&timestamp, b + offset, sizeof(uint64_t));
    return timestamp;
}

double
convert_to_unix(uint64_t packet_timestamp) {
    int seconds = int(double(packet_timestamp) / PER_SECOND);
    int useconds = (double(packet_timestamp) / PER_USECOND) - double(seconds) *
    return BASE_TIMESTAMP + seconds + useconds / 1.0e6;
}
```

- Base_timestamp is based on LA's time zone
- Different time zone will not filter correctly

mi2log File Upload

- Currently, we only support uploading mi2log files in the same directory as MobileInsight-Cloud.
  - It's not feasible to get the file absolute path when uploading files.
  - Limit the ability to upload files remotely.
- We can get the mi2log object and its filename. But current MobileInsight does not support loading bytes object as input.
  - The OfflinePlayer class only accept file path as input.
  - Takes about 30mins to upload 1095 files (~200MB).
  - Duplicate file operation.
    - First upload the file to get the filename.
    - Then open the file to process data.

mi2log File Download

- Similar issue with mi2log file upload.
- Short-term solution:
  - Load the Redis's data to a tmp.mi2log file for buffering.
  - Perform filtering on tmp.mi2log.
  - Save the tmp.mi2log file as the desired filename

Future Work

- Process the timestamp based on local setting.
- Implement a loading/saving function that directly takes a byte object as input/output.
  - Prevent duplicate file operations.
  - Enable upload/download operations on remote server.
    - Necessary for deploying to the cloud.
- Scale up to do further performance analysis.

Q & A