MobileInsight Cloud
Cloud Service and Web Interface of MobileInsight

Chun-Yao Chang
cyc1028@cs.ucla.edu
UCLA
Los Angeles, CA, USA

Lily Lin
lilyl3@g.ucla.edu
UCLA
Los Angeles, CA, USA

Yi-Chun Lo
yichunlo0919@ucla.edu
UCLA
Los Angeles, CA, USA

Po-Nien Kung
ponienkung@ucla.edu
UCLA
Los Angeles, CA, USA

ABSTRACT

MobileInsight is a software tool enabling real-time cellular network monitoring and smartphone analytics. It provides open access to detailed 5G protocol information, allowing researchers and developers to quickly and easily obtain low-level network data through user-friendly APIs. Despite its applicability, certain limitations limit its usage to a broader group of users. MobileInsight only operates on local machines and cannot share logs with other users. Its challenging installation process further raises the barriers to using this tool. In this project, we aim to solve the limitation by proposing MobileInsight Cloud, a cloud version of MobileInsight that can filter, upload, and download logs from a client machine without the need to download MobileInsight. Our framework includes a Redis database that stores key/value pairs with in-memory database architecture, allowing rapid retrieving from the database. At the end of the report, we further show the filtering time of our method, demonstrating its efficiency.

ACM Reference Format:
Chun-Yao Chang, Lily Lin, Yi-Chun Lo, and Po-Nien Kung. 2024. MobileInsight Cloud Cloud Service and Web Interface of MobileInsight. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The ever-evolving landscape of cellular network technology, particularly with the advent of 5G, demands robust real-time network monitoring and analysis tools. To address this need, MobileInsight [1] emerged as a powerful software tool designed for in-depth cellular network monitoring and smartphone analytics. By providing open access to comprehensive 5G protocol information, MobileInsight helps researchers and developers swiftly and conveniently access low-level network data through its user-friendly APIs. This capability is critical for various applications, from network performance optimization to developing next-generation mobile applications. While MobileInsight has proven to be a powerful tool, its limitations have limited its full potential. One of the primary constraints is that MobileInsight operates solely on local machines, limiting its ability to share logs with other users or systems. This localized operation model not only limits collaborative efforts but also imposes significant logistical challenges for users who need to aggregate and analyze data across multiple devices or locations. However, with the introduction of MobileInsight Cloud, we are opening up new avenues for collaboration and data sharing, empowering each member of the research and development community to contribute to the collective knowledge. To overcome these limitations, we propose an innovative solution: MobileInsight Cloud. This cloud-based iteration of MobileInsight aims to enhance the accessibility and functionality of the original tool by introducing capabilities for filtering, uploading, and downloading logs directly from a client machine. Crucially, this can be achieved without the necessity of installing MobileInsight locally. The core of our framework is a Redis database that employs an in-memory data architecture, enabling rapid retrieval and storage of key/value pairs. This architectural choice ensures that users can quickly access and manipulate network data, significantly improving the efficiency of network monitoring and analysis tasks. In this report, we present a comprehensive overview of MobileInsight Cloud, detailing its architecture, implementation, and the specific enhancements it offers over the original MobileInsight tool. Specifically, for efficient log management, MobileInsight Cloud utilizes Redis as the log storage database. It ensures rapid data access and processing with in-memory database architecture, and the key-value storage system provides high scalability, making it an optimal choice for managing large data volumes with speed and ease. By transitioning to a cloud-based solution and incorporating Redis, MobileInsight Cloud can become a more user-friendly and robust tool for cellular network monitoring and analytics. In Figure 1, we show the pipeline of our MobileInsight framework. When a client uploads a log file to MobileInsight Cloud, the system parses it into JSON format and stores both the original log file and the JSON file in Redis. If the client needs to perform analysis or filtering, MobileInsight Cloud can retrieve the log and JSON files from Redis, execute the required operations, and then allow the client to download the processed mi2log or JSON file to their machine. Besides the step-by-step implementation details, we also provide empirical evidence of the

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Figure 1: An example of our MobileInsight Cloud framework.

system's performance, particularly focusing on the filtering time of our method. By demonstrating the efficiency and practicality of MobileInsight Cloud, we aim to highlight its potential as a valuable resource for the broader research and development community engaged in cellular network technology. We provide our project code here: https://github.com/ChunYaoChang/CS219.

2 DATA TRANSFORM

To facilitate the efficient handling and visualization of mobile network logs, MobileInsight Cloud employs a robust data transformation process. This process converts raw mi2log files into JSON objects and stores them in a Redis database. JSON objects are advantageous for display purposes due to their structured format and ease of manipulation.

2.1 Conversion and Storage

The raw mi2log log files are transformed into JSON objects using the following approach:

- The key for each JSON object in Redis is constructed as: `f'{filename}:{type_id}:{timestamp}:{order}'`. This key schema provides enough information to identify and retrieve log entries without loading the entire dataset, allowing for efficient filtering and access.
- The value stored in Redis is the log file in JSON format. This structure enables easy display and processing of log data.

2.2 Extending the Analyzer Class

To process mi2log format data, the Analyzer class in the MobileInsight library is extended. The enhancements include:

- Parsing the raw XML string in the "Msg" field to an XML object, enabling structured access to the log message contents.
- Adding an "order" field to preserve the sequence of log items, ensuring that the logs are displayed and analyzed in the correct order.

2.3 Storing Raw Files

In addition to converting mi2log files to JSON objects, the raw mi2log files are also stored. This allows users to download the original mi2log files if needed for further analysis or archival purposes.

By implementing these data transformation steps, MobileInsight Cloud ensures that log data is efficiently processed, stored, and retrievable, enhancing the overall user experience and enabling detailed analysis of mobile network performance.

3 DATA FILTERS

Originally, MobileInsight supports time-based and type_id filtering. Expanding on this functionality, we now add timestamp-based filtering. Users can specify a start_timestamp and end_timestamp to filter logs within this time interval. To enable this feature, we modify MobileInsight's APIs which requires recompilation. For detailed recompilation instructions, please refer to the README.md in our repository. In the following subsections, we trace through our modified code.

3.1 offline_replayer.py

MobileInsight utilizes the OfflineReplayer class to load log files. Within this class, the enable_log method enables the monitoring of log with specific type_ids.

- `def enable_log(self, type_name)`

To enable timestamp filtering within a user-specified range, we modify the enable_log method by adding start_timestamp and end_timestamp parameters to its function header to define the filtering range. By default, these parameters are set to None to maintain the original type_id filtering functionality.

- `def enable_log(self, type_name, start_timestamp = None, end_timestamp = None)`

enable_log does not perform filtering instead it passes start_timestamp and end_timestamp to dm_collector_c.set_filtered. To retrieve the filtered log, OfflineReplayer calls dm_collector_c.receive_log_packet.

3.2 dm_collector_c.cpp

We modified dm_collector_c_set_filtered to parse start_timestamp and end_timestamp from the arguments tuple and pass them to manager_set_filter. The passing start_timestamp and end_timestamp will be set to 0 and infinity by defalut, which means there is no timestamp argument passing into it. In dm_collector_c_receive_log_packet function, it calls the manager_export_binary to retrieve the binary packet data, then decode it to Python object. We continue tracing the export_manager module.

3.3 export_manager.h

The original packet state structure does not contain timestamp interval information. We modify the structure ExportManagerState by adding in two members: start_timestamp and end_timestamp. These two variables are used for filtering, and the value will be set by the manager_set_filter function in dm_collector_c.cpp.

```
struct ExportManagerState {
    FILE * log_fp;
    std::string filename;
    std::set<int> whitelist;
    double start_timestamp;
    double end_timestamp;
  }
```

3.4 export_manager.cpp

The filtering logic is implemented inside export_manager.cpp.

3.4.1 manager_set_filter. manager_set_filter is a new method that we added to set the values of start_timestamp and end_timestamp in ExportManagerState.

3.4.2 manager_export_binary. This is the core part of the new feature. Originally, manager_export_binary only filters log based on type_id. To support filtering by packet timestamp, we modify the condition as follows:

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

To filter packets within a user-defined time range, it is essential to know each packet's timestamp. This timestamp is extracted from its binary log file using two newly defined functions:

3.4.3 get_timestamp. As specified in log_packet.h, a packet's timestamp is located at the 8-byte offset within a packet header. The get_timestamp function retrieves this *raw* timestamp.

3.4.4 convert_to_unix. As mentioned previously, the get_timestamp function returns the raw timestamp of a packet. To convert this raw timestamp into UNIX format, we adopt the conversion process specified by _decode_by_fmt_modem function in log_packet.cpp.

Note that this conversion is computed based on BASE_TIMESTAMP, manually set to 315993600, which will lead to time zone limitation. This we will discuss further in the LIMITATION section.

4 MOBILEINSIGHT CLOUD

MobileInsight Cloud is a powerful platform designed to facilitate the collection, analysis, and visualization of mobile network data. It provides users with an intuitive interface to manage and interpret mobile network logs, enabling detailed insights into network performance and behavior. Powered by Streamlit[1], a Python library for web applications, MobileInsight Cloud is developed entirely in pure Python, requiring no front-end experience. It is easy to deploy and benefits from a strong open-source community.

[1]https://streamlit.io/

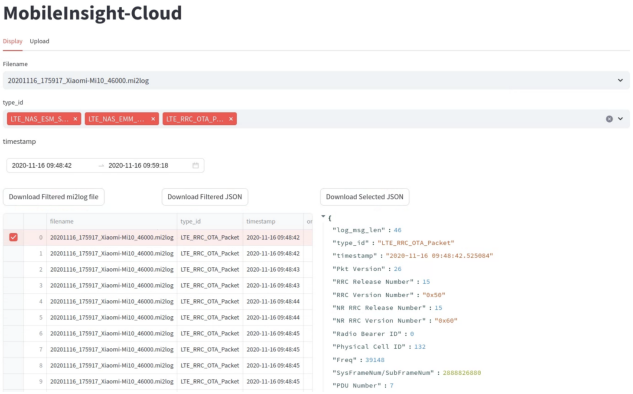

Figure 2: Display Tab of MobileInsight Cloud

4.1 Display Tab

The Display tab is the primary interface for interacting with mobile network logs. It consists of three main sections:

(1) **Filters**: Users can filter logs based on three criteria:
- **Filename**: Select the specific log file you wish to examine.
- **Type ID**: Choose the type of log messages to display (e.g., LTE_NAS_ESM, LTE_NAS_EMM, LTE_RRC_OTA_Packet).
- **Timestamp**: Define a time range to narrow down the logs to a specific period.

(2) **Log List**: A tabulated list of log items appears in the lower left part of the display, showing:
- **Filename**: The name of the log file.
- **Type ID**: The type identifier for the log.
- **Timestamp**: The time the log was recorded.
- **Order**: The sequential order of the logs.

Users can click on individual log items to view their detailed information.

(3) **Log Detail**: The selected log's details are visualized as a JSON object in the lower right part of the display.
Additionally, the interface provides buttons for downloading the logs:
- **Download Filtered mi2log File**: Downloads the logs based on the applied filters.
- **Download Filtered JSON**: Downloads the filtered logs in JSON format.
- **Download Selected JSON**: Downloads the JSON object of the selected log item.

A progress bar is displayed during download operations to indicate the progress.

4.2 Upload Tab

The Upload tab allows users to upload mi2log files to the Redis database. This functionality is crucial for storing and later analyzing log data. A progress bar is shown during the upload process, providing real-time feedback on the status of the upload.

MobileInsight Cloud thus offers a comprehensive solution for managing mobile network logs, from uploading raw log files to

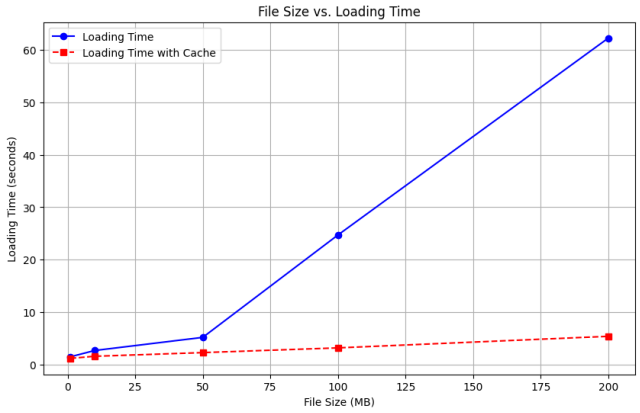

Figure 3: Comparison of File Loading Times with and without Cache

filtering and downloading processed log data. Its user-friendly interface ensures that users can efficiently navigate through large datasets and extract meaningful insights.

5 PERFORMANCE ANALYSIS

5.1 Data Load Time

Figure 3 illustrates the relationship between file size (in MB) and loading time (in seconds). The blue line represents the loading time without cache, while the red dashed line represents the loading time with cache. The data shows that using cache significantly reduces the loading time, especially for larger file sizes. That's because Streamlit provides a caching mechanism that can significantly reduce load times for expensive computations or data-loading operations. Streamlit uses the `@st.cache_data` and `@st.cache_resource` decorators to cache the results of function calls. When a function decorated with `@st.cache_data` is called, Streamlit checks if it has already been executed with the same parameters. If so, it retrieves the result from the cache instead of re-executing the function.

5.2 Data Filter Time

Using Streamlit's cache mechanism can also be beneficial for filtering operations on large pandas DataFrames. When filtering data, especially if the DataFrame is large, it can be computationally expensive. Caching the results of these filter operations can significantly speed up the Streamlit app. However, as previous mentioned, Streamlit checks if it has already been executed with the same parameters. When user uploads the mi2log file with same filename but different content, Streamlit is not able to detect the difference and reload the data. Thus, it still require further investigation and discussion to improve the caching efficiency and correctness.

6 LIMITATION

Despite the robust functionalities offered by MobileInsight Cloud, there are certain limitations.

6.1 Timestamp

Recall that we need to convert the timestamp into UNIX format after extracting it from the binary packet. This conversion is based on the manually computed BASE_TIMESTAMP, which aligns with the Los Angeles time zone. Therefore, if the packet timestamp is in a different time zone, packets may be filtered incorrectly.

6.2 mi2log File Upload

Currently, the system only supports uploading mi2log files that are located in the same directory as MobileInsight Cloud. This restriction is due to the challenge of obtaining the absolute file path when uploading files, which limits the ability to upload files remotely.

Key limitations in the upload process include:

- **File Path Requirement**: The current implementation does not support loading bytes objects as input. The `OfflinePlayer` class in MobileInsight only accepts a file path as input.
- **Duplicate File Operations**: The process involves duplicative steps where the file is first uploaded to obtain the filename and then opened again to process the data. This redundancy leads to inefficiency.
- **Upload Duration**: Uploading 1095 files, approximately 200MB in total, can take about 30 minutes, indicating a need for optimization in handling large volumes of files.

6.3 mi2log File Download

Similar challenges are faced during the download of mi2log files. The short-term solution to address these issues involves using a temporary file for buffering and filtering operations.

Key limitations in the download process include:

- **Temporary File Usage**: The current workaround involves loading Redis data into a temporary file (tmp.mi2log) for buffering. Filtering operations are performed on this temporary file before saving it as the desired filename.
- **Efficiency Concerns**: This method, while functional, is not optimal and may introduce inefficiencies due to the additional steps involved in handling temporary files.

These limitations highlight the need for further enhancements in the MobileInsight Cloud platform to streamline file handling processes and improve overall efficiency. Future updates should aim to support remote file uploads, optimize data processing, and reduce redundancies in file operations.

7 FUTURE WORK

While MobileInsight Cloud provides a solid foundation for collecting, analyzing, and visualizing mobile network data, there are several areas for future enhancement to improve functionality and performance.

7.1 Enhanced Timestamp Processing

To ensure the accuracy and relevance of log data, future versions of MobileInsight Cloud will process timestamps based on local settings. One possible method is to calculate the time difference between the local setting and Los Angeles, and add or subtract the value to the BASE_TIMESTAMP. This will allow users to view logs

in their local time zone, enhancing the interpretability and usability of the data.

7.2 Direct Byte Object Handling

Implementing loading and saving functions that directly accept byte objects as input and output will streamline the file handling process. This enhancement will prevent duplicate file operations, making the system more efficient by eliminating the need for intermediate steps.

7.3 Remote Server Operations

To enable seamless integration with cloud infrastructure, MobileInsight Cloud will support upload and download operations on remote servers. This capability is crucial for cloud deployment, allowing users to manage log files from any location without being restricted to local directories.

7.4 Performance Optimization

Scaling up the platform to handle larger datasets efficiently is a key area of focus. Future work will involve optimizing the performance of the system to handle high volumes of data swiftly and accurately. This will include:

- Enhancing the data processing algorithms to reduce upload and download times.
- Improving the efficiency of data storage and retrieval mechanisms.
- Implementing advanced filtering techniques to quickly access relevant log entries without loading entire datasets.

7.5 Scalability and Cloud Deployment

Further performance analysis and optimization will be conducted to ensure that MobileInsight Cloud can scale effectively. This includes:

- Enhancing the infrastructure to support concurrent processing of multiple log files.
- Leveraging cloud-native technologies to improve scalability and resilience.
- Conducting thorough performance testing to identify and address potential bottlenecks.

By addressing these areas, MobileInsight Cloud will continue to evolve, offering users a more powerful and flexible tool for mobile network analysis. These enhancements will ensure that the platform remains robust and capable of meeting the growing demands of mobile network data analysis.

REFERENCES

[1] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. Mobileinsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking* (New York City, New York) *(MobiCom '16)*. ACM, New York, NY, USA, 202–215. https://doi.org/10.1145/2973750.2973751