CS 211:

Demystifying the Mobility of Cellular IoT

Harrison Cassar, Ricky Guo, Vedant Sathye

Mentor: Jinghao Zhao

Introduction

Harrison Cassar
1st-Year Masters
Computer Science
harrisoncassar@cs.ucla.edu

Ricky Guo
1st-Year Masters
Computer Science
rickyrguo@cs.ucla.edu

Vedant Sathye
1st-Year Masters
Computer Science
vsathye@g.ucla.edu

Outline

1. Background and Motivation
2. Methods
3. Datasets and Experiments
4. Results
5. Conclusions and Future Work

Background and Motivation

Background

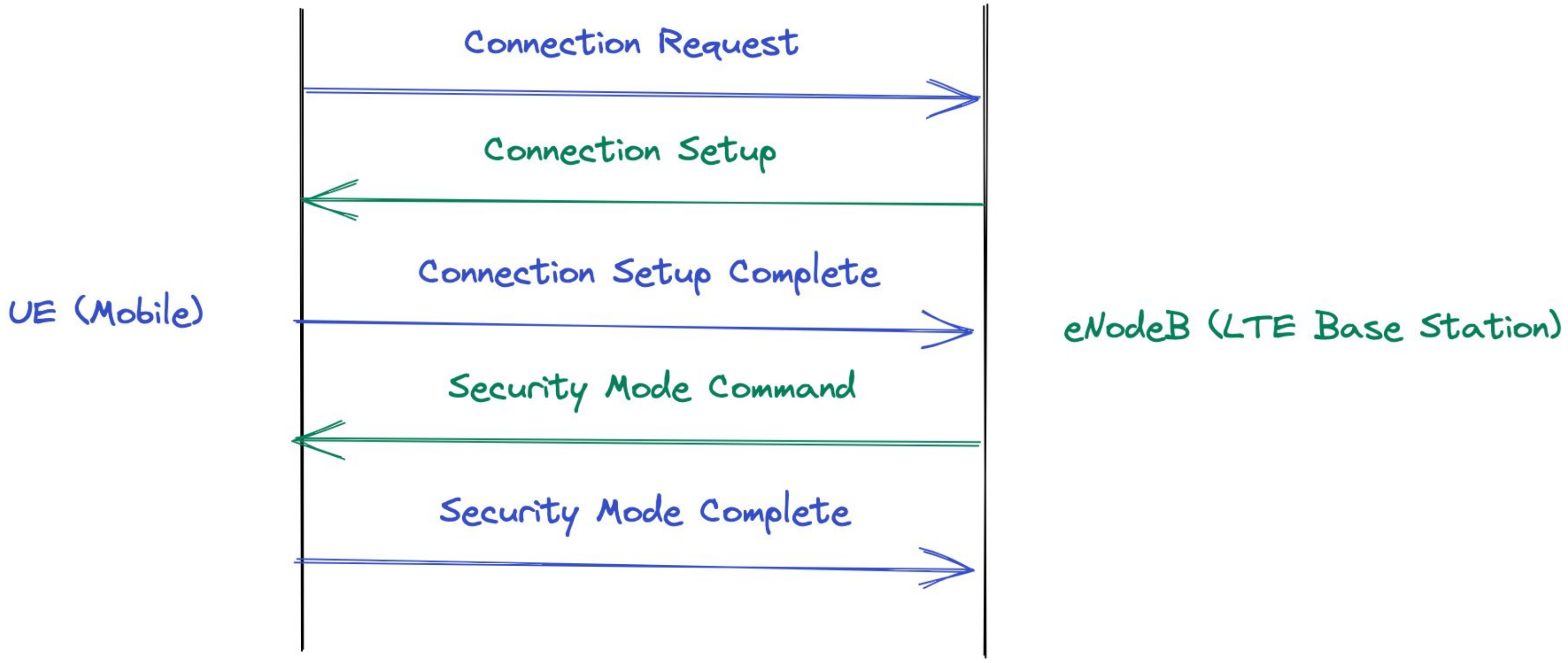
High-Level Summary

- C-IoT device association process with a network.
  - Understanding the attach signaling procedures.
  - Understanding handover signaling procedures.

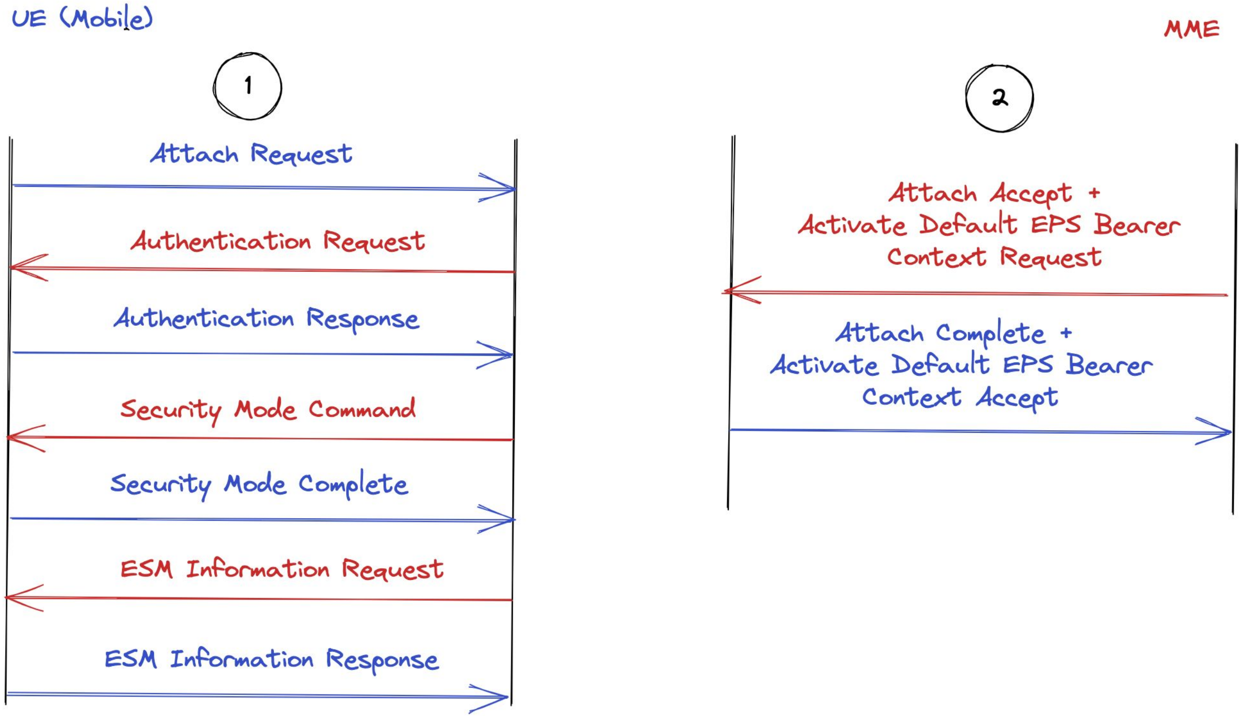
Challenges

- Finding documentation on the C-IoT signaling procedures!
- Not easy to read/understand!
  - Some difference from 5G/LTE. Even harder to map to the data packets that we have.

RRC Attach Procedure

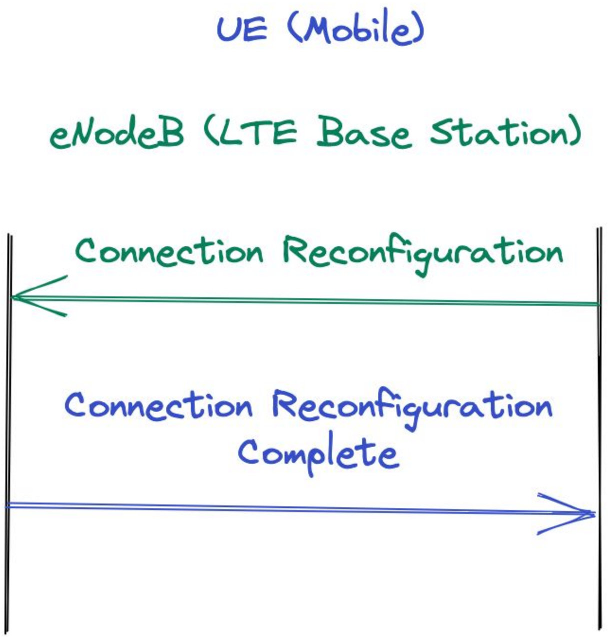


NAS Attach Procedure

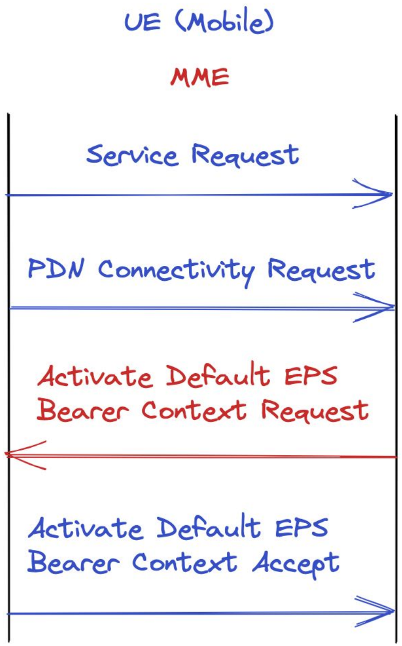


Connection Setup

RRC (Initial)

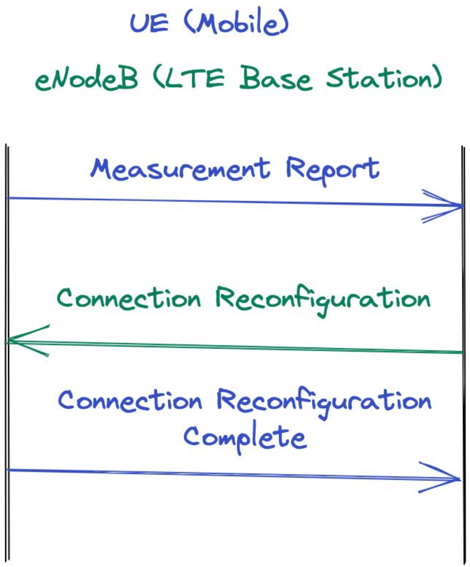


NAS (Idle State Reconnection)

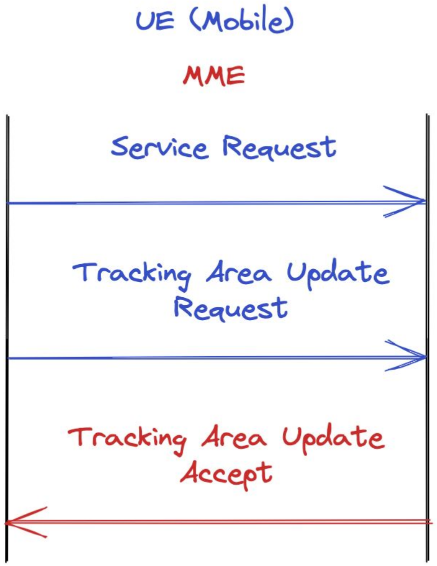


Handover Procedures

Base Station Handover



MME Handover



Motivation

Desire

"Any-time, anywhere" service for CIoT

- Seamless data transmission
- Maintain little-to-no user intervention
- Independent of user mobility, location

Problem

Mobility-induced issues for mobile networks still exist for CIoT!

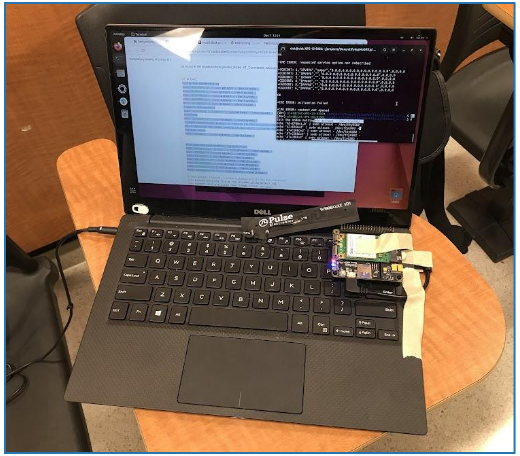
- Spatial-temporal dynamics, problematic for everyone
  - Signaling overhead to maintain service (operator)
  - Poor service and/or disruption during handover, location updates (users)

Problem: high latency → inconvenient for us!

Methods

Testbed Setup

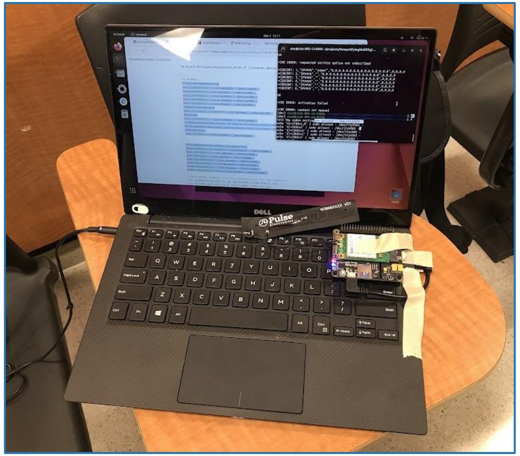
- **Hardware:** Laptop + External CIoT Module
  - Laptop
    - Ubuntu 22.04.1 LTS, 64-bit
    - Dell Inc. XPS13 9350, Intel i5-6200U
  - External CIoT Module
    - Telit ME910C1-WW LTE Cat M1/NB1
    - RPi 3G/4G & LTE Sixfab Base HAT
    - T-Mobile SIM Card



- **Software:** MobileInsight + atinout + Python
  - MobileInsight enables mobile network monitoring
  - `atinout` enables AT commanability
  - Python scripts for configuring CIoT module, generating ping traffic, and collecting traces

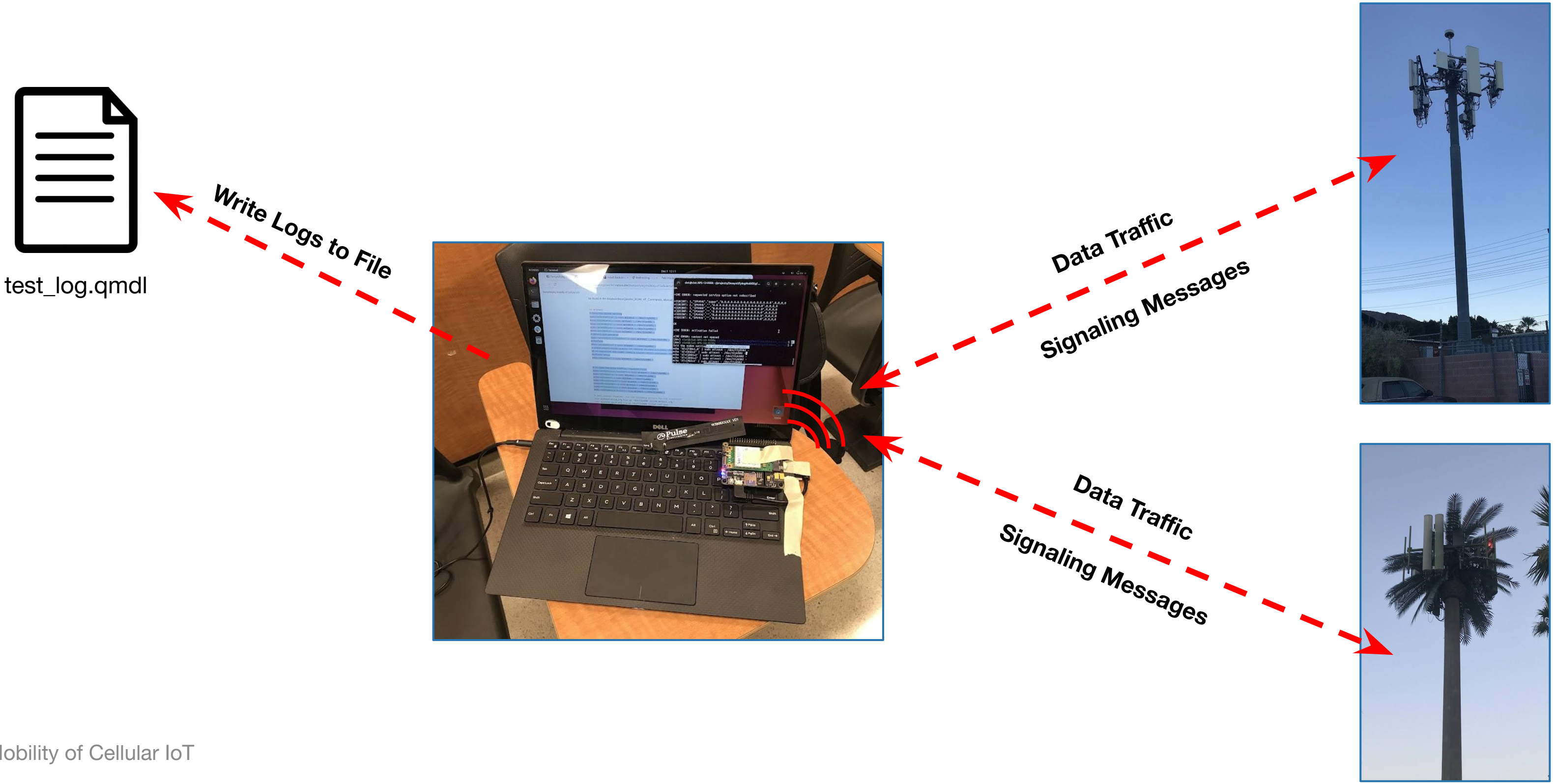
Data Collection Procedure

Summary of Steps



1. Power-on testbed machine
2. Connect CIoT module via USB
3. Initialize and configure module via:
    a. Bunch of AT commands to init modem, select operator, register to the network, etc.
    b. Performing test ping to ensure connection
    c. Writing a device-specific configuration file
4. Run Python scripts to:
    a. Collect all mobility traces over USB
    b. Generate ping traffic to incur handover events
5. *Move!*

Data Collection (visually)

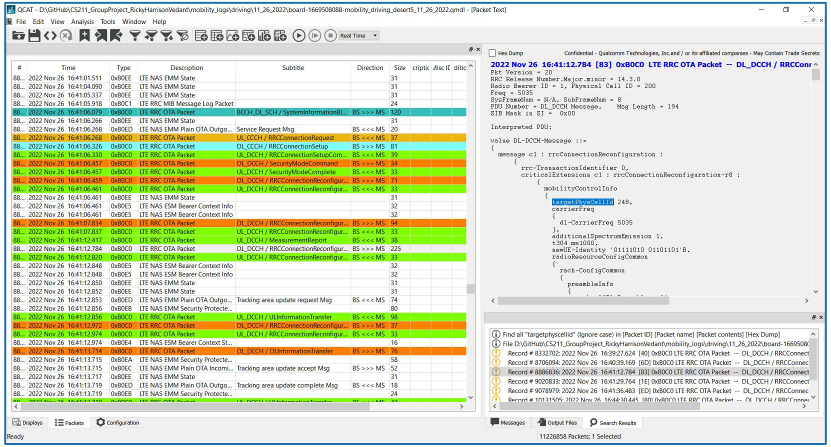


Data Collection Challenges

- **Testbed setup**
  - Local machine compatibility issues
    - M1 Macbook Pro and MobileInsight
    - Windows 10 with Ubuntu subsystem dependencies installation issues
  - Module unknown unresponsiveness
  - Faulty physical USB serial connection
- **Data collection**
  - Determining where BS boundaries are
    - Triggering handover events
  - Procedure sophistication
    - e.g. No ping traffic → little-to-no handovers
  - Corrupted log files
    - Likely from off-nominal file closure

Data Processing Setup

- **Input Data**
  - QMDL Files
- **Tools**
  - QCAT
    - Windows application
  - Mobile Insight
    - Offline Replayer
    - Buffer Analyzer



Sample QCat Log

Data Processing Setup (cont.)

High-Level Summary

- Defining key KPI latency metrics
  - Use the time deltas between different key signaling events.
- Data Processing with Python
  - QMDL files are not directly readable.
  - Use Mobile Insight Python library (Offline Replayer & Buffer Analyzer) to run logging data to process.

Challenges

- Modeling state transitions can be quite complex.
- Trading off the simplicity (along with the maintainability and readability) of the script with the ability to model more complex transitions.

Data Processing Setup (cont.)

```python
if BufferAnalyzer.has_nas_activate_default_eps_bearer_context_request_record(message):
    self.nas_logging_silencer('[8.2a/11a/3b] Parsing nas activate default eps bearer context request record.')
    self.nas_message_timestamps['activate_default_eps_bearer_context_request'] = decoded_message['timestamp']
    if self.nas_previous_state == NASState.PDN_CONNECTIVITY_REQUEST and self.nas_previous_previous_state != NASState.SERVICE_REQUEST:
        self.nas_split_latency_metrics['active_pdn_connectivity_request_to_activate_default_eps_bearer_context_request'].append(
            BufferAnalyzer.get_timedelta_millis(self.nas_message_timestamps['pdn_connectivity_request']
            - self.nas_message_timestamps['activate_default_eps_bearer_context_request'])
        )
    elif self.nas_previous_state == NASState.PDN_CONNECTIVITY_REQUEST and self.nas_previous_previous_state == NASState.SERVICE_REQUEST:
        self.nas_split_latency_metrics['idle_pdn_connectivity_request_to_activate_default_eps_bearer_context_request'].append(
            BufferAnalyzer.get_timedelta_millis(self.nas_message_timestamps['pdn_connectivity_request']
            - self.nas_message_timestamps['activate_default_eps_bearer_context_request'])
        )
    else:
        print('Invalid state transition from {} to {}'.format(
            self.nas_previous_state, NASState.ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST
        ))
    self.nas_previous_state, self.nas_previous_previous_state, self.nas_previous_previous_previous_state = NASState.ACTIVATE_DEFAULT_EPS_BEARER_CONTEXT_REQUEST, self.nas_previous_state, self.nas_previous_previous_state
```

Handling State Transition Taking into Account Previous Two States

Data Processing Setup (cont.)

Implementation Status

- Captured abridged [happy] path of RRC and NAS procedures.
  - Room to expand on looking into failed signaling.
- Can dig deeper into non-latency metrics.

Findings

- Positive
  - Signaling procedures are normally prescriptive.
- Negative
  - Difficult to draw the line on just *how much* detail in latency we want.
  - Sparsity of data due to case analysis.

General Pipeline

Setup Testbed → Collect Logs and Traces → Parse/Process Raw Data → Perform Downstream Task

Setup Testbed

- Laptop + External CIoT Module
- Connect to Network

Collect Logs and Traces

- *Move! Move!*
- Induce interesting CIoT mobility-related events, such as handover

Parse/Process Raw Data

- Filter relevant logs
- Extract key KPI Metrics

Perform Downstream Task

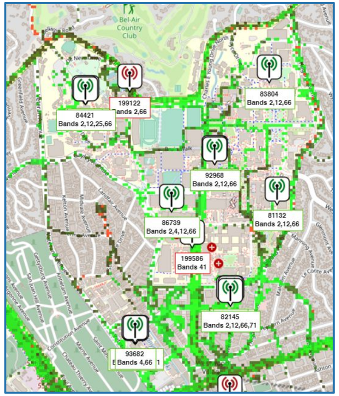
- Visualize
- Do something else!
    - …such as handover prediction!

Datasets and Experiments

Raw Datasets

- **Two mobility scenarios**
    - Driving (40-70mph)
        - Around Palm Springs (relatively urban)
        - To/from LA to Palm Springs
    - Walking (<4mph)
        - Around UCLA campus

KEY: In all data considered, numerous handover and non-handover events

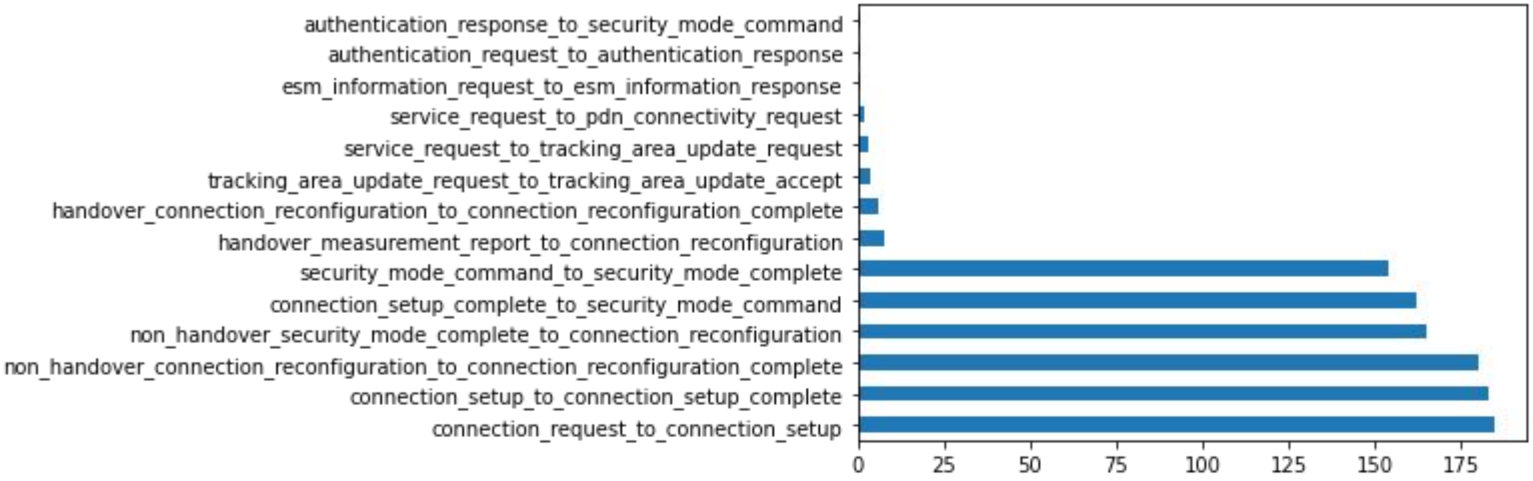


T-Mobile B.S. Map Around UCLA/Westwood

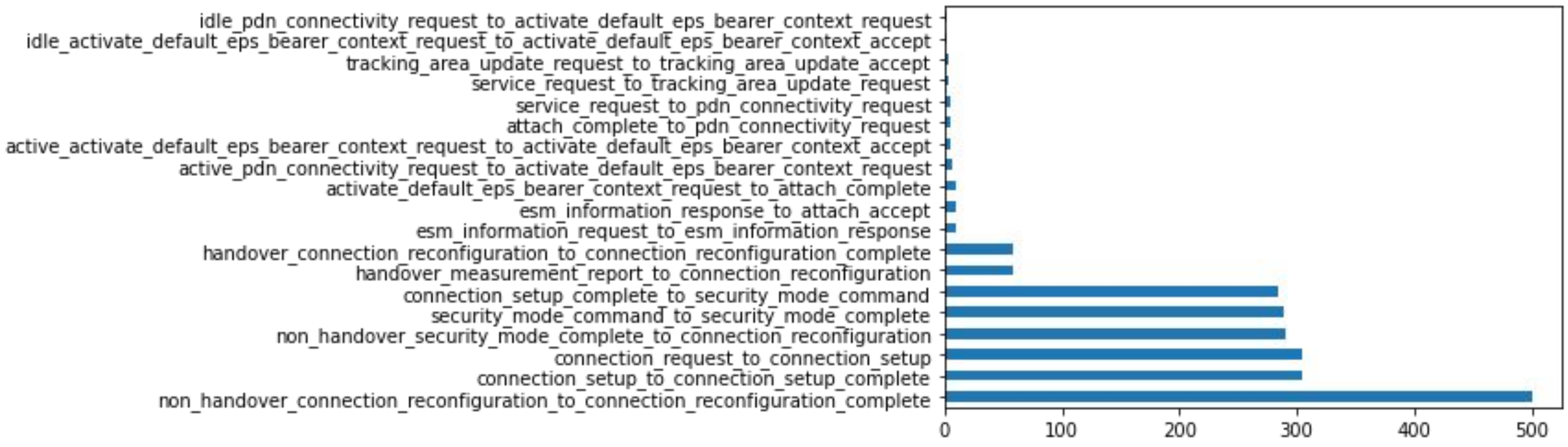
Pauley Pavilion at UCLA

Results

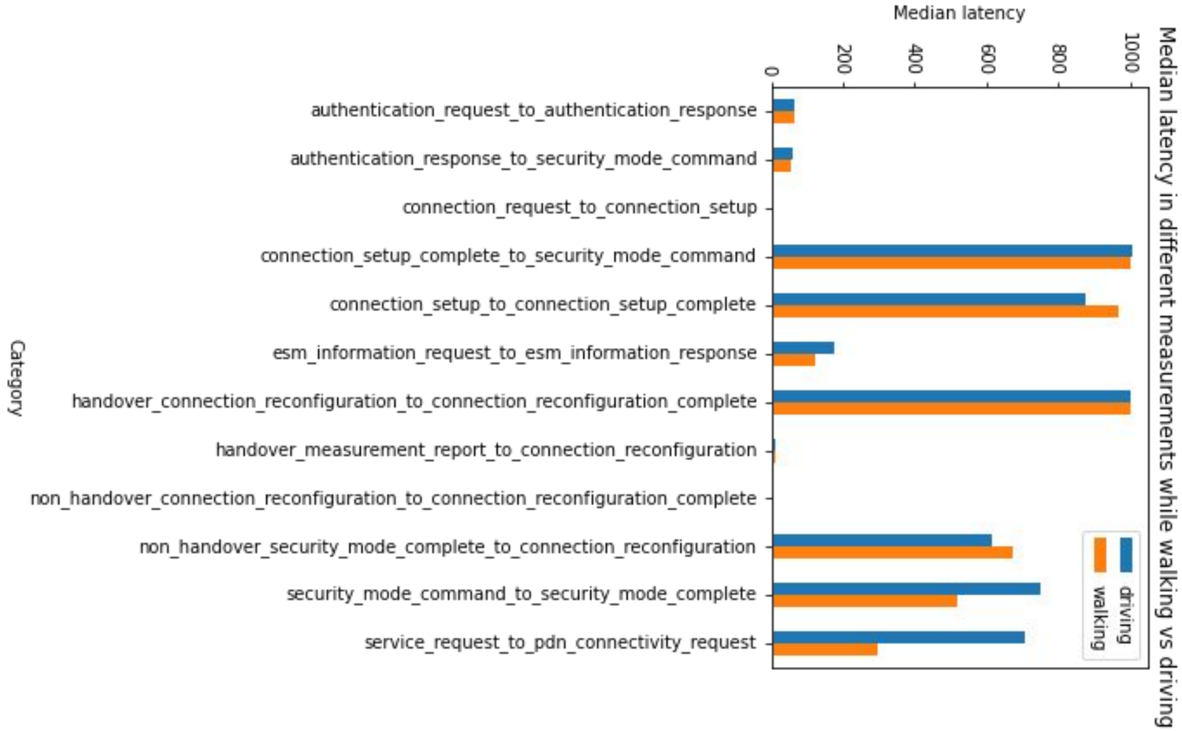
Driving message counts



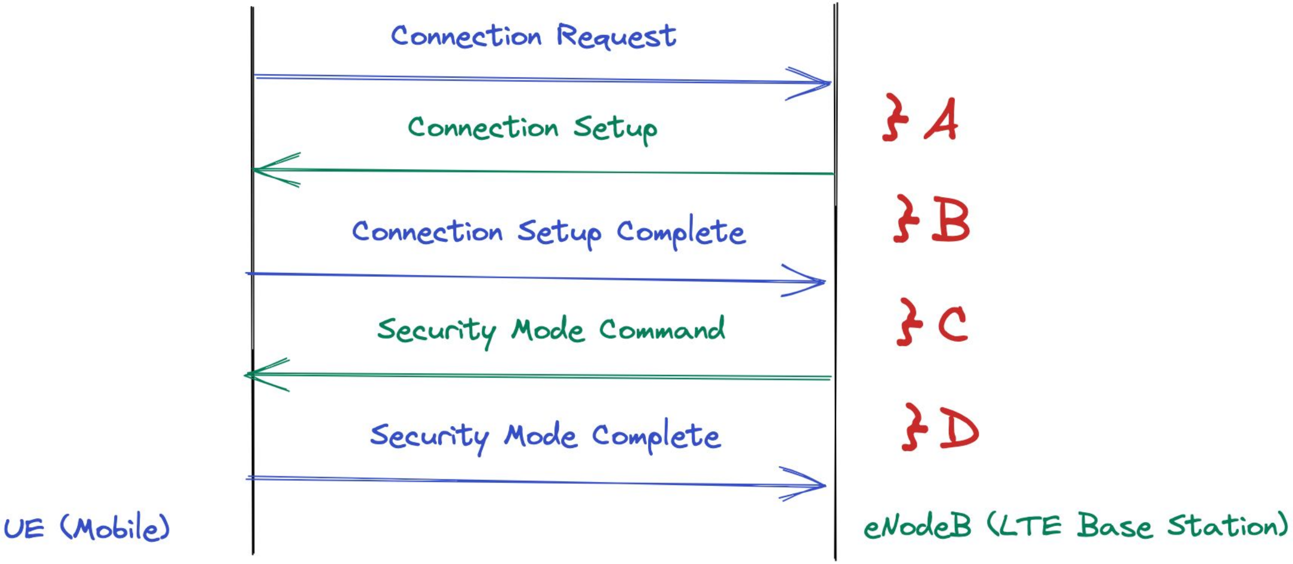
Walking message counts



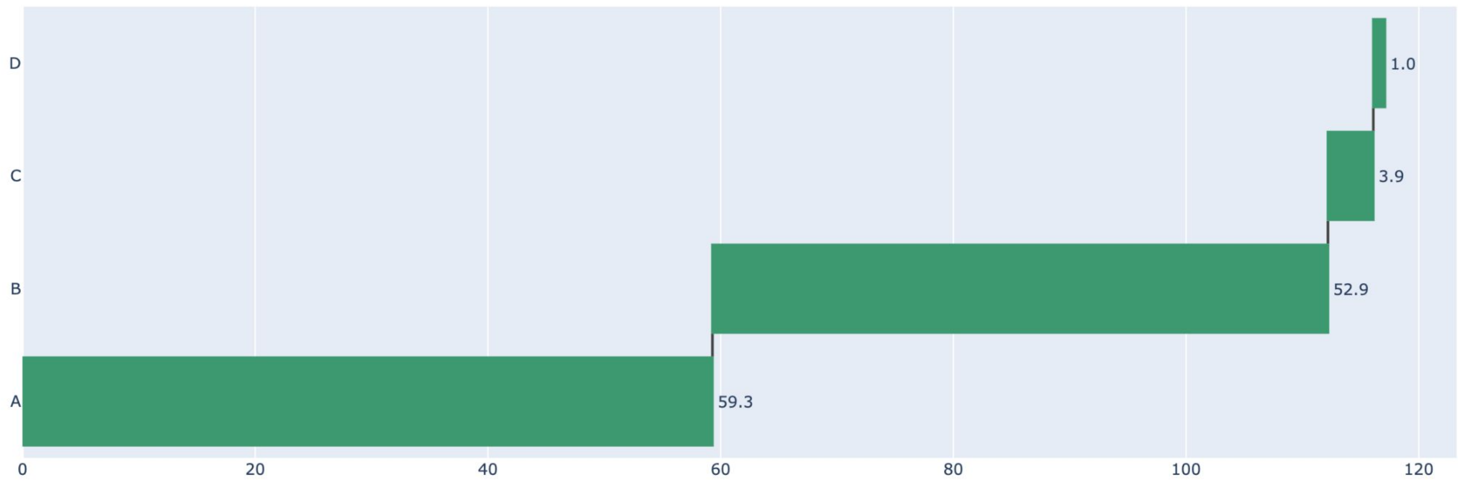
Median latencies for walking and driving



RRC Attach


Connection Request
Connection Setup
Connection Setup Complete
Security Mode Command
Security Mode Complete
} A
} B
} C
} D
UE (Mobile)
eNodeB (LTE Base Station)

Median Latency Waterfall for RRC Attach

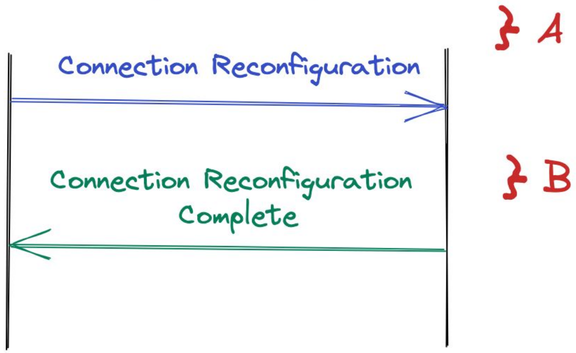


RRC Setup

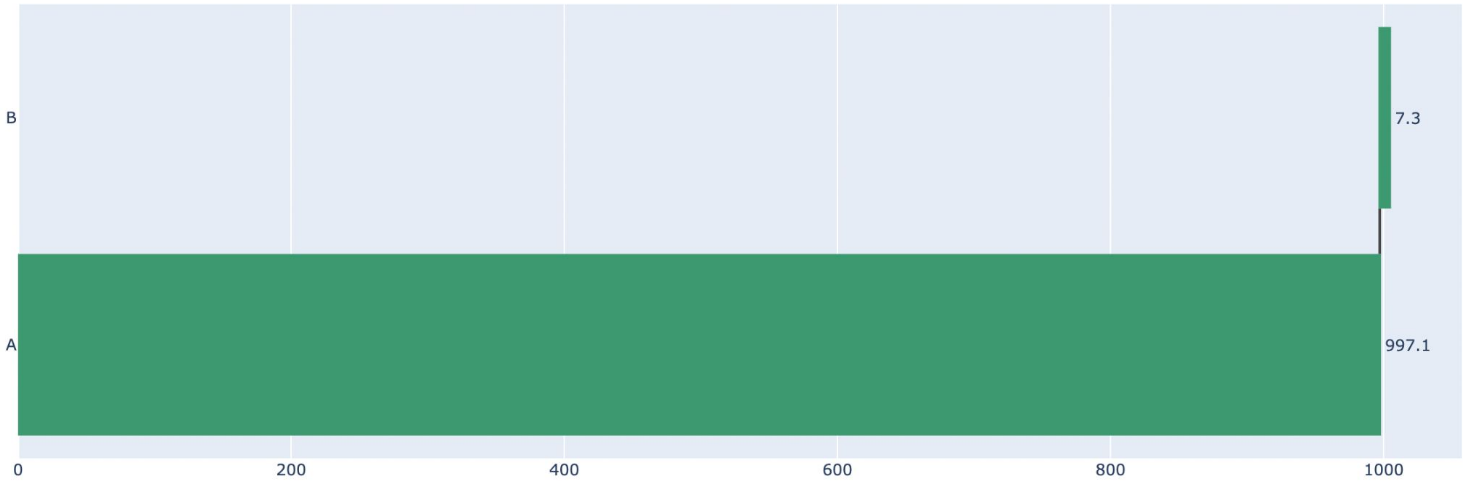
UE (Mobile)

eNodeB (LTE Base Station)

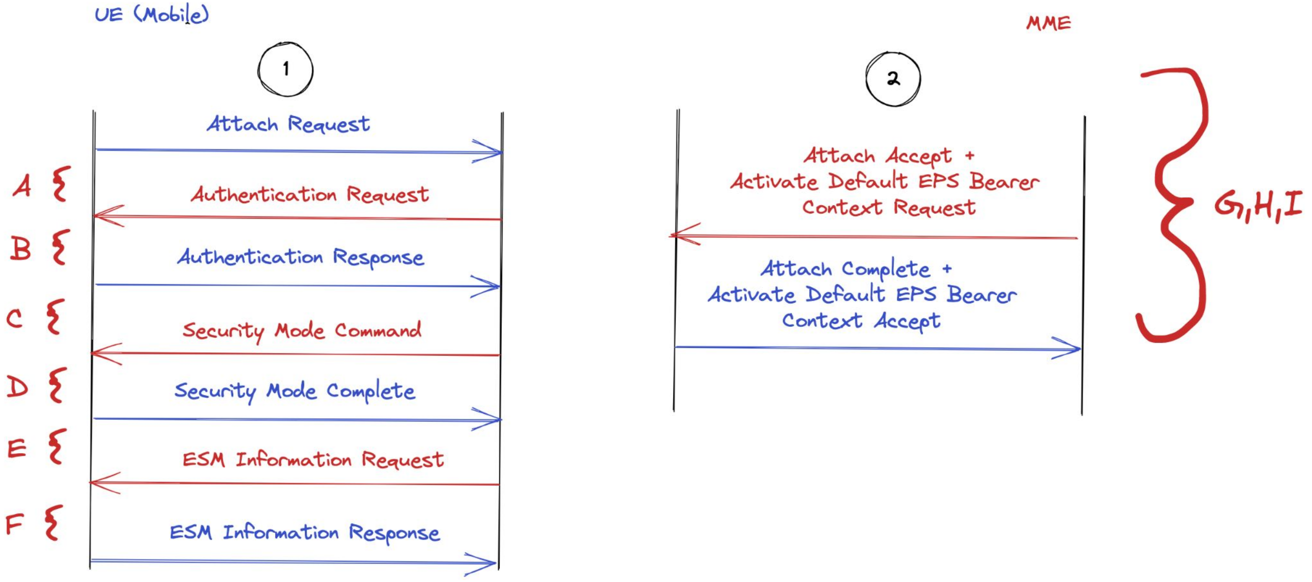
Security Mode Complete...



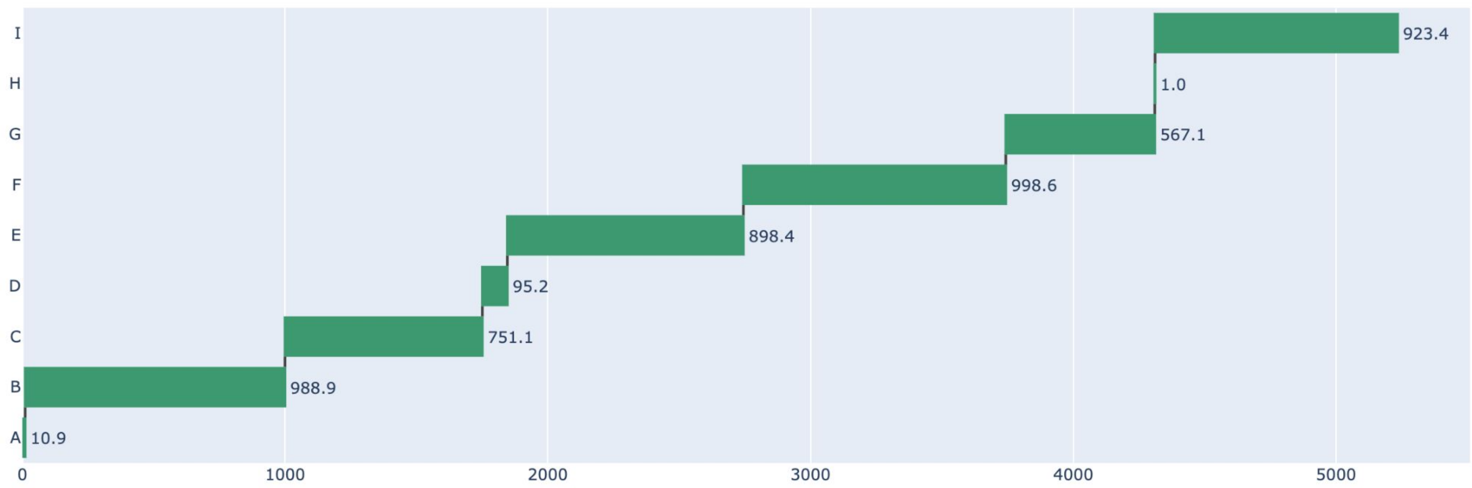
Median Latency Waterfall for RRC Setup



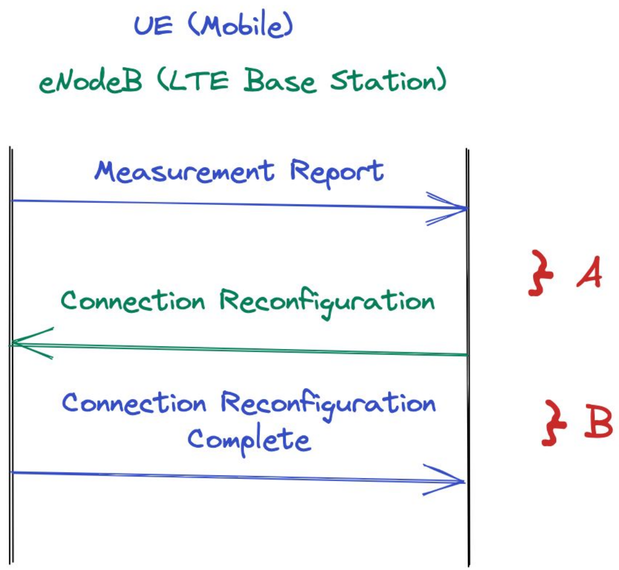
NAS Attach

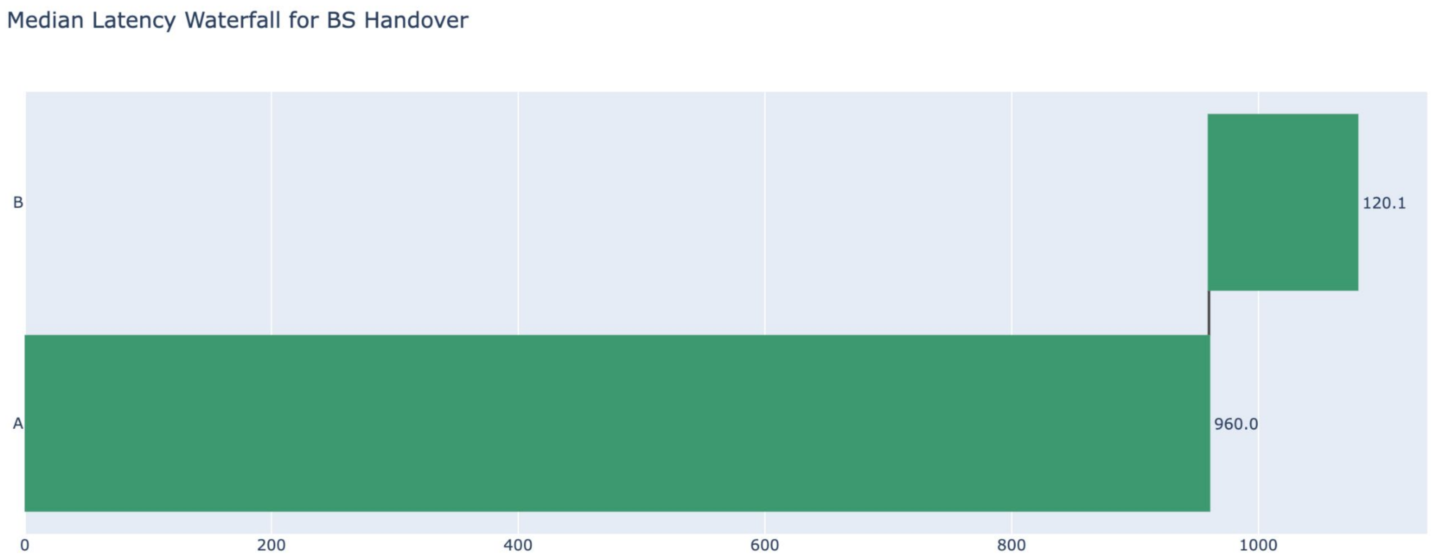

UE (Mobile)
MME
1
Attach Request
A
Authentication Request
B
Authentication Response
C
Security Mode Command
D
Security Mode Complete
E
ESM Information Request
F
ESM Information Response
2
Attach Accept +
Activate Default EPS Bearer
Context Request
Attach Complete +
Activate Default EPS Bearer
Context Accept
G,H,I

Median Latency Waterfall for NAS Attach

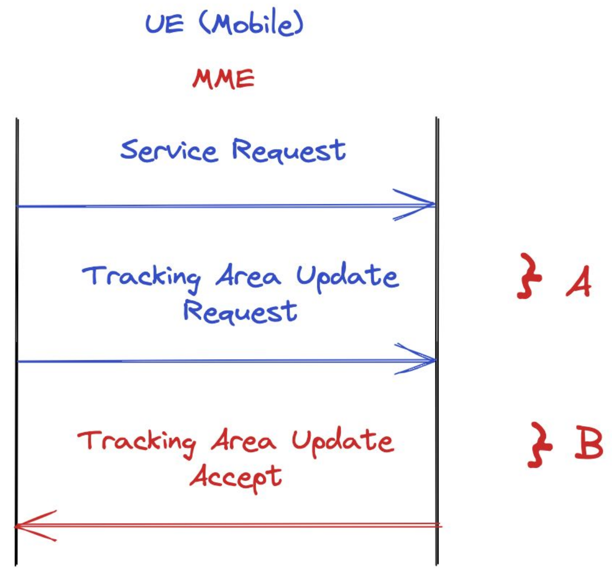


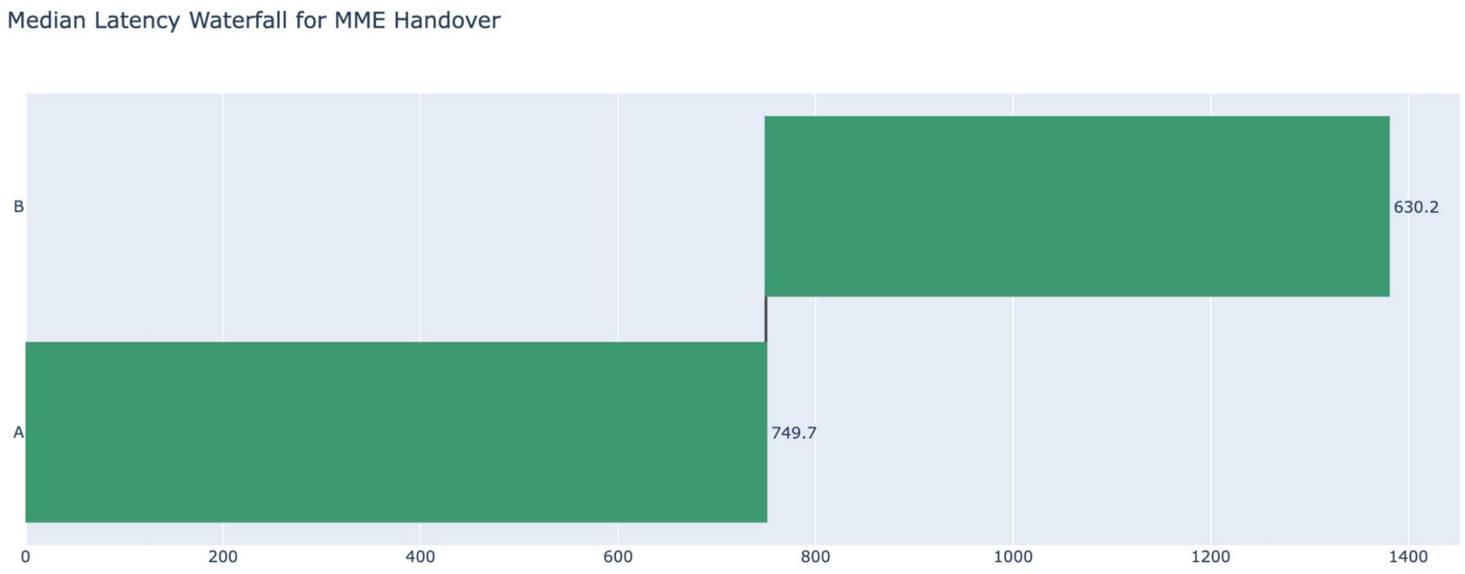
BS Handover


UE (Mobile)
eNodeB (LTE Base Station)
Measurement Report
Connection Reconfiguration
A
Connection Reconfiguration Complete
B

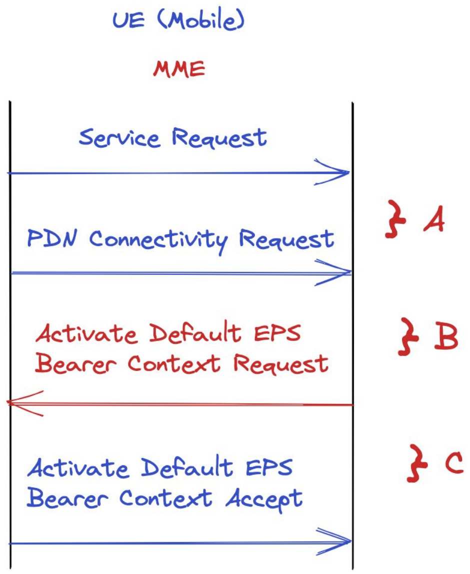

Median Latency Waterfall for BS Handover
B
A
120.1
960.0
0
200
400
600
800
1000

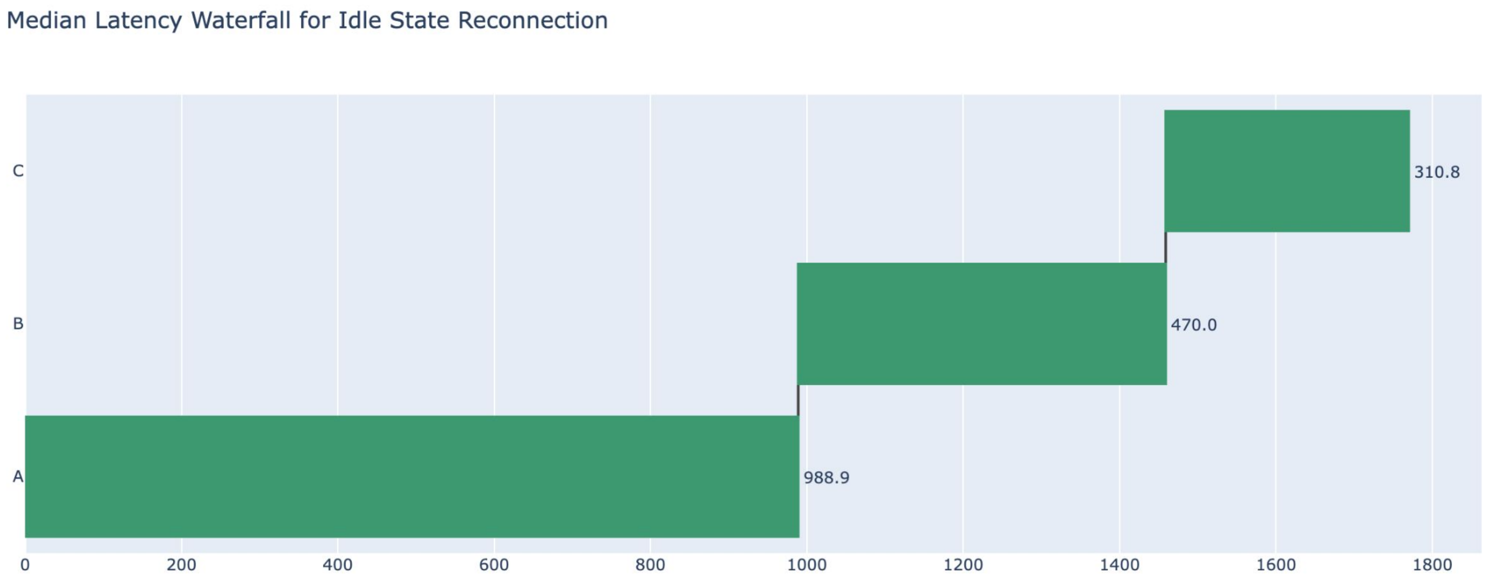
MME Handover



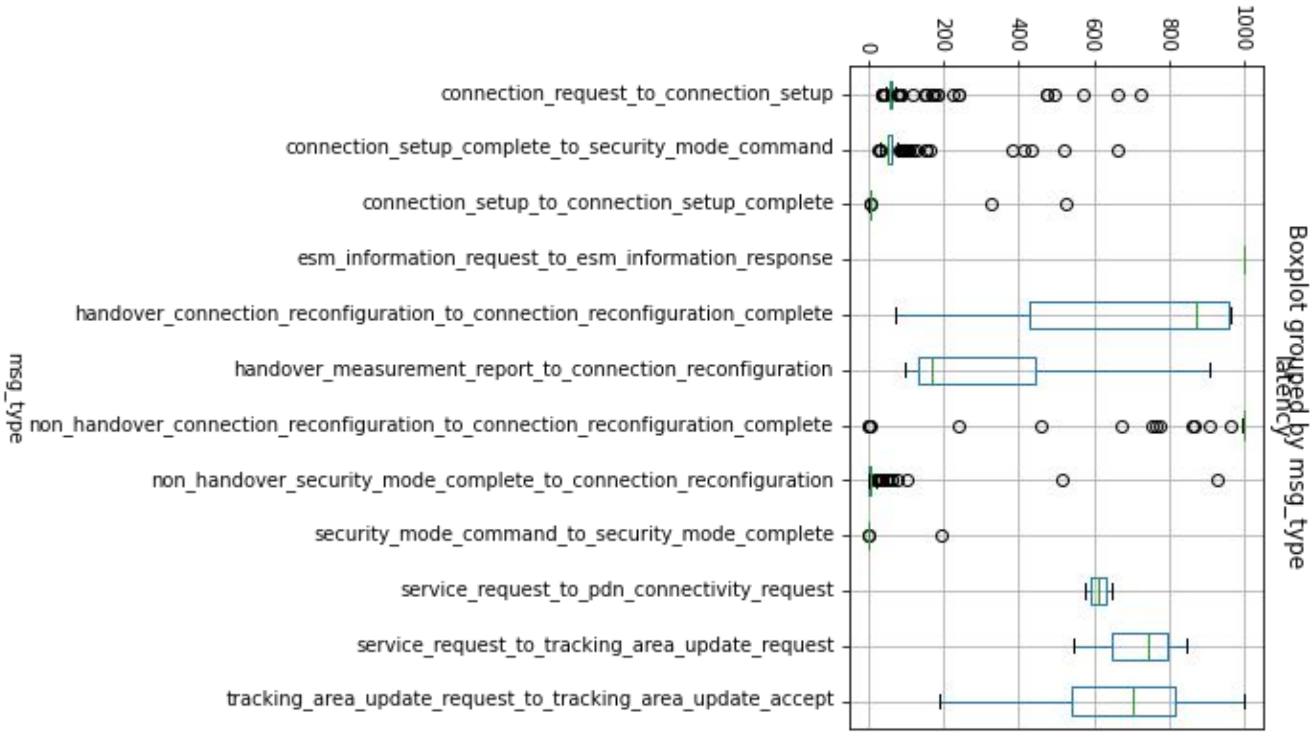

Median Latency Waterfall for MME Handover
B
A
630.2
749.7
0
200
400
600
800
1000
1200
1400

Idle State Reconnection

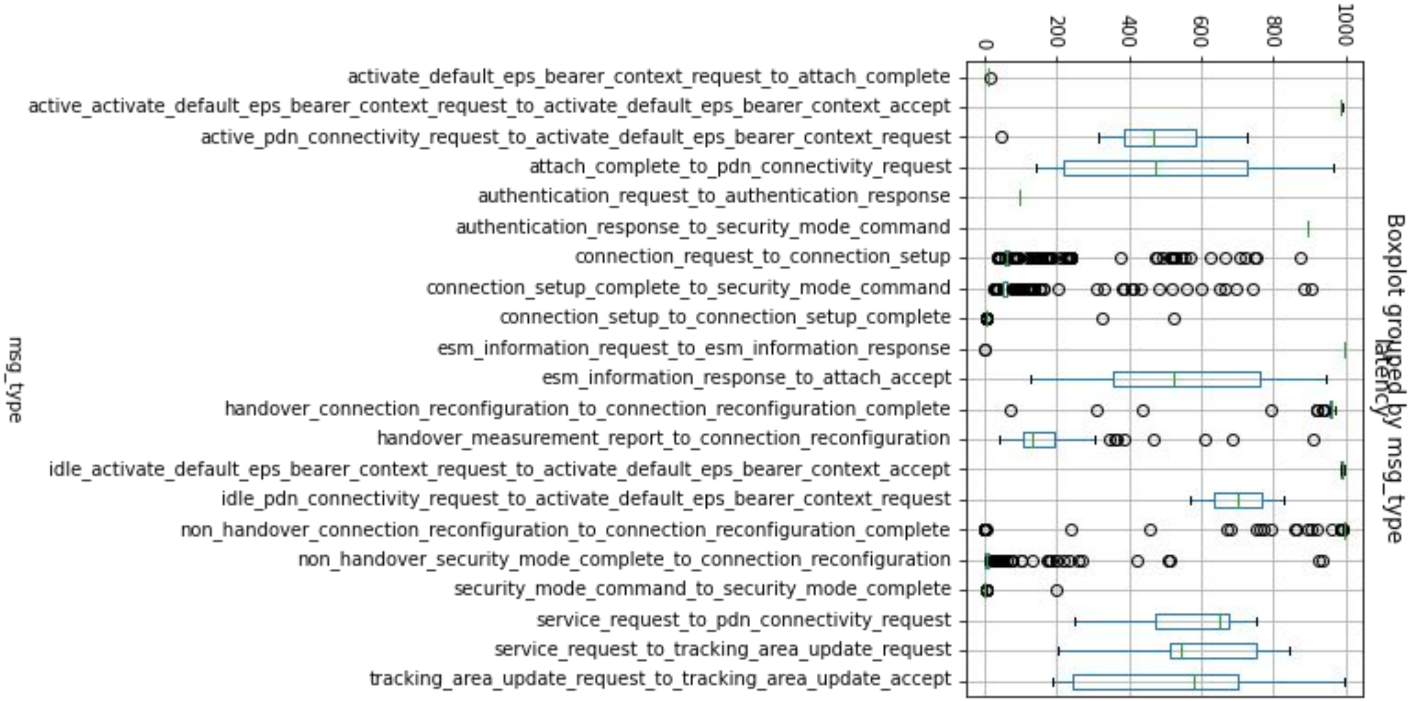



Median Latency Waterfall for Idle State Reconnection
C
B
A
310.8
470.0
988.9
0
200
400
600
800
1000
1200
1400
1600
1800

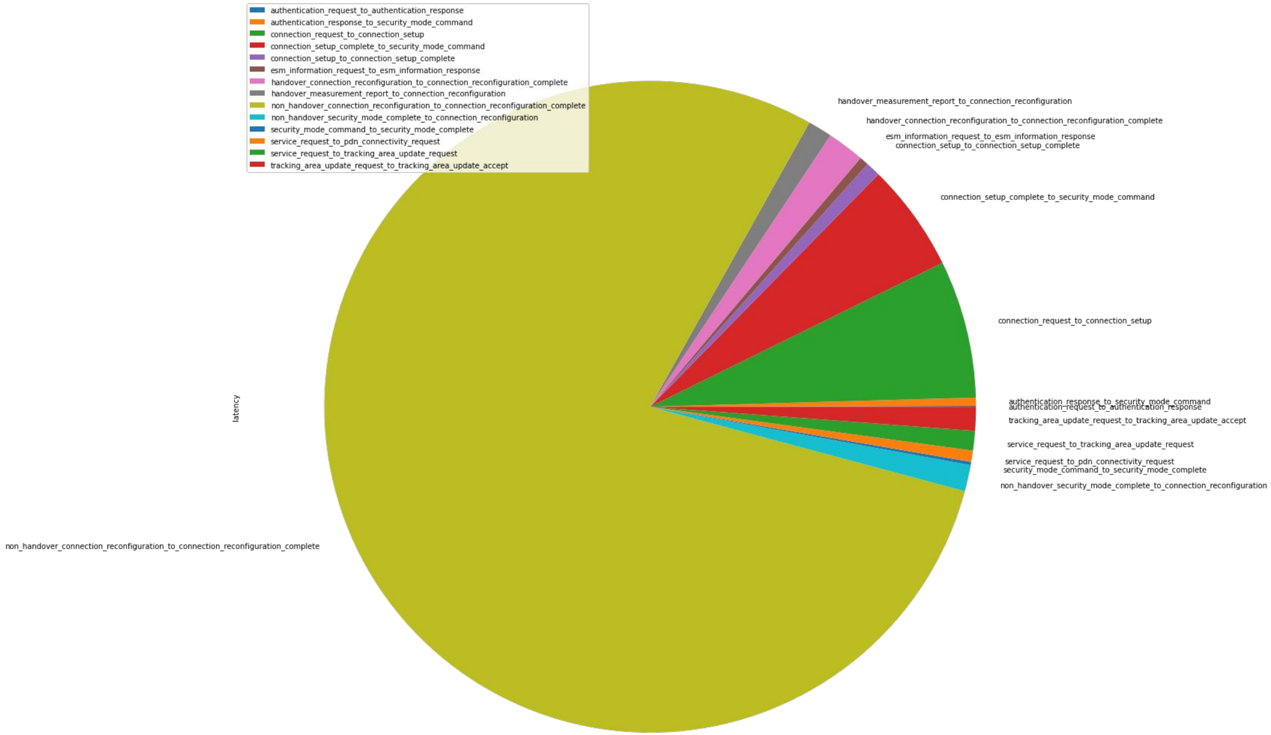
Driving latencies



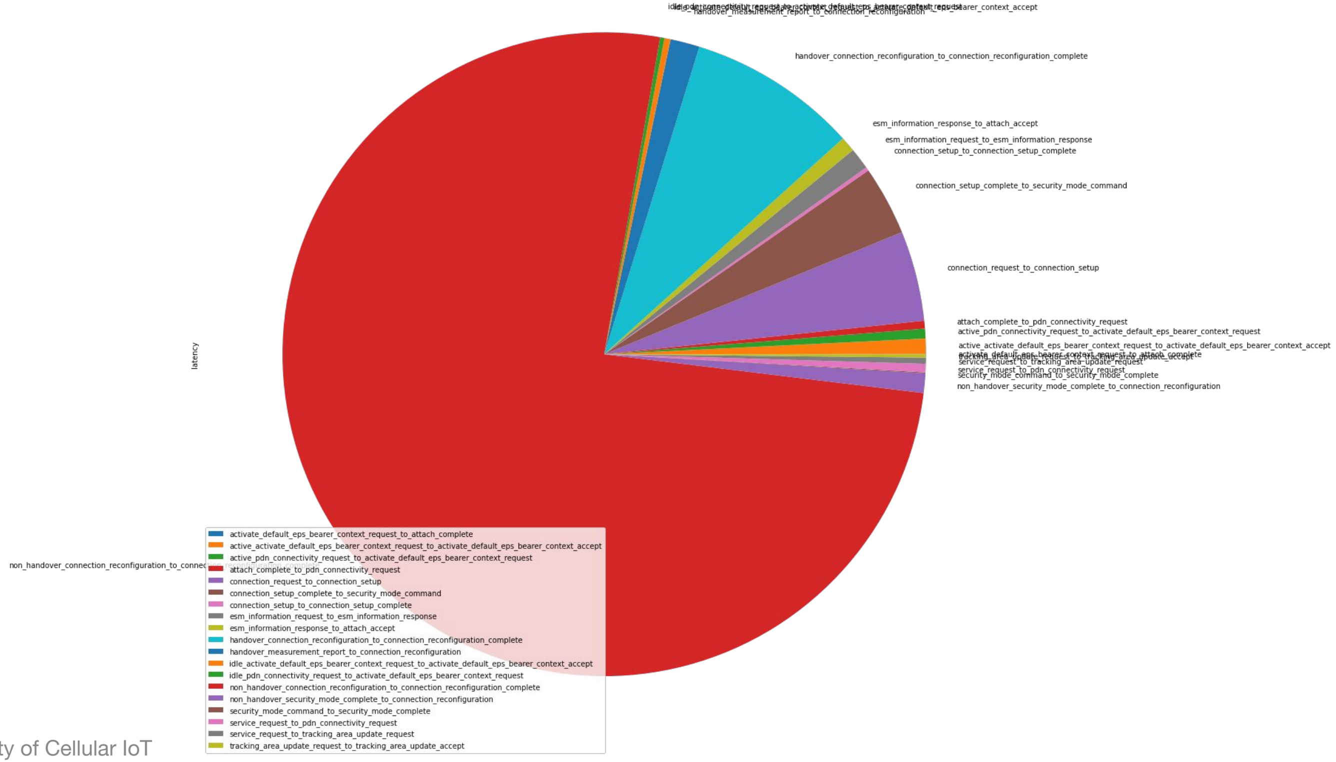
Latency Distributions



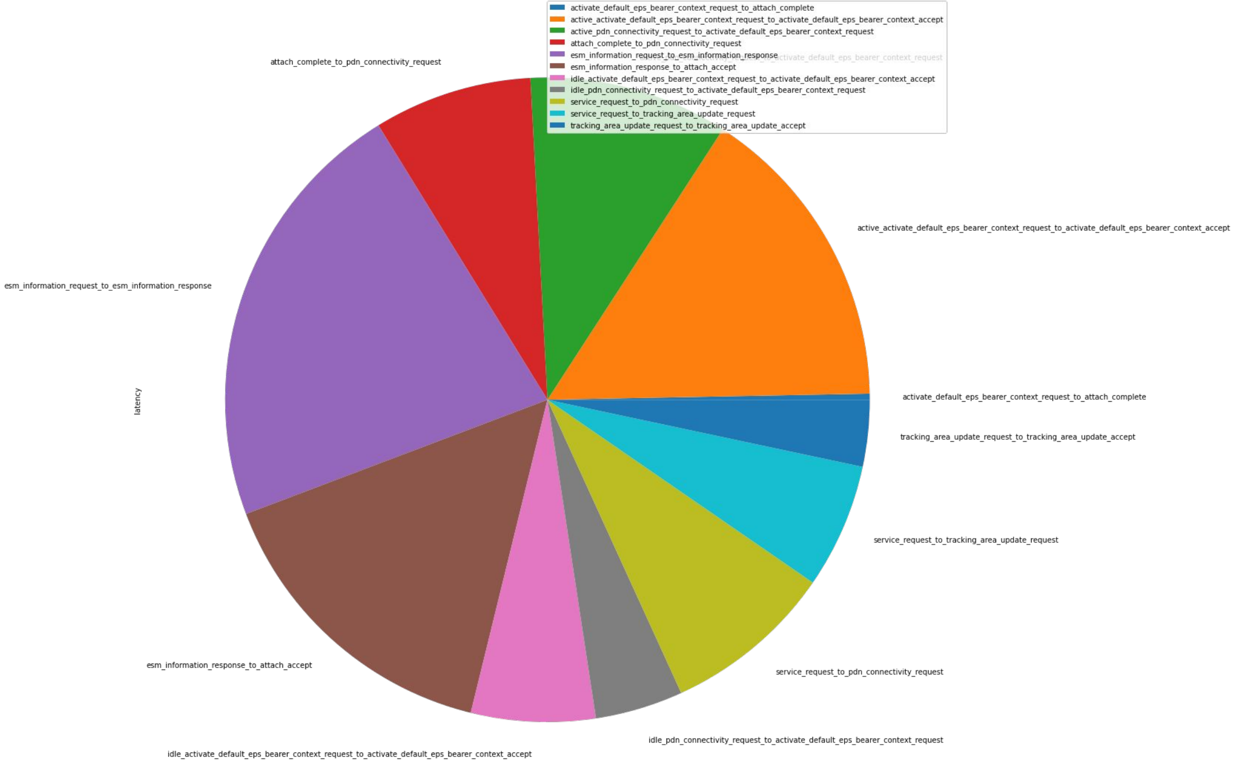
Driving latency time distribution



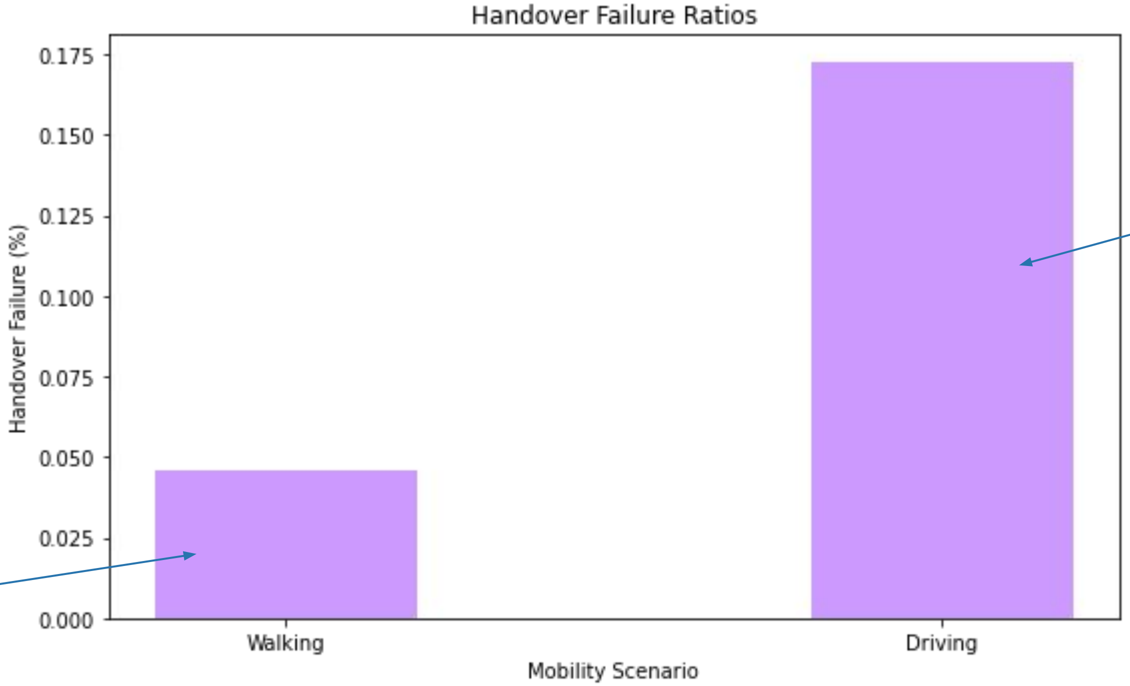
Walking latency time distribution



Walking NAS time distribution



CIoT Handover Failure Rates



62 successes, 3 failures (~0.046% failure)

24 successes, 5 failures (~0.172% failure)

…an almost 4x difference!

Conclusions and Future Work

Conclusions

- Minimal apparent latency difference between CIoT mobility scenarios
  - A few metrics have slightly less for walking
- Handover-related signaling latency significantly higher than for similar non-handover signaling messages
  - Matches expectation
  - Handover bottleneck: RRC's Connection Reconfiguration process
- Handover failure ratio higher for driving mobility
  - Driving had ~3.7x failure rate than walking
  - Perhaps due to mobility-induced issues, or leaving new B.S. ranges

Future Work

- Collect greater diversity of data
  - Case studies in different data sources/mobility scenarios, may influence our handover and latency metrics/conclusions
- Explore more robust testbed setups/procedures/scripts
  - Avoid scrapping of data due to mobility-problems and setup issues
- Interesting downstream tasks
  - Handover prediction using ML
    - Perhaps extraction of more key KPIs
  - Spatial-temporal latency estimation and/or prediction
- Explore reasons behind failed handovers
  - Could enable handover prediction task
- Comparison of IoT to non-Iot bottlenecks in similar procedures

Special Thanks

- Special thanks to our mentor, **Jinghao Zhao**, who has not only made us excited about this project’s content, but also helped us tirelessly throughout this project’s development in many different ways.

Thank you!