CS 219:

Cloud-Based IoT Management with SIM Card

Harrison Cassar, Ricky Guo,
Disha Zambani, Albert Stanley
Mentor: Jinghao Zhao

CS 219
Cloud Computing

Introduction

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

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

Disha Zambani
1st-Year Masters
Computer Science
dmzambani@cs.ucla.edu

Albert Stanley
1st-Year Masters
Computer Science
albertstanley@ucla.edu

Outline

1. Background and Problem
2. System Overview and Implementations
    a. Issues/Challenges
    b. Current Status and Limitations
3. Demo
4. Insights Learned and Future Work

Background and Problem Statement

Background and Motivation

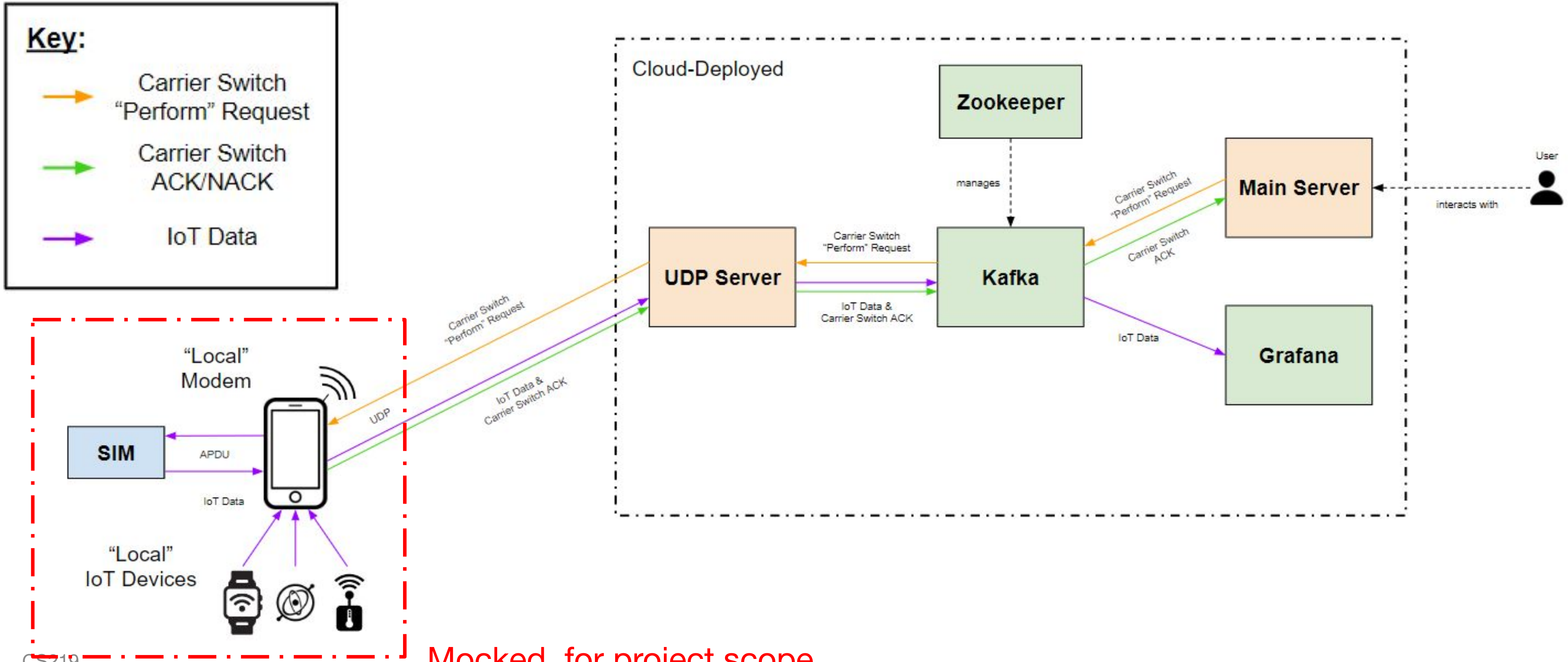
- 5G IoT enables multi-carrier capabilities with 5G SIM cards.
- SIM cards themselves are able to transmit packets containing IoT data from IoT devices connected to the SIM card device to the cloud.
- This opens up a need to create a cloud service to manage SIM cards and the IoT data sent over.
    - Data on the cloud to be used for downstream tasks (i.e. analytics, machine learning)

Problem and Project Goals

- We would like to create services that achieve the following:
  - Create a 2-way message transmission service that is able to send IoT data to and receive messages from the cloud.
    - IoT data visualizations with Grafana.
  - Use the above cloud service to enable message queues for incoming IoT data for future downstream tasks.
    - Kafka for high throughput of real-time data feeds and allows for messaging, storage, and stream processing.
  - Develop a service that can control and change the SIM card carrier with a user-friendly UI.
    - Carrier switch status updates.
    - Lightweight Flask server (Python) with HTML and CSS.
- We want to achieve this by having little-to-no modifications on the SIM card device itself.
  - By having the SIM card communicate directly with the cloud we avoid any modifications on the device.

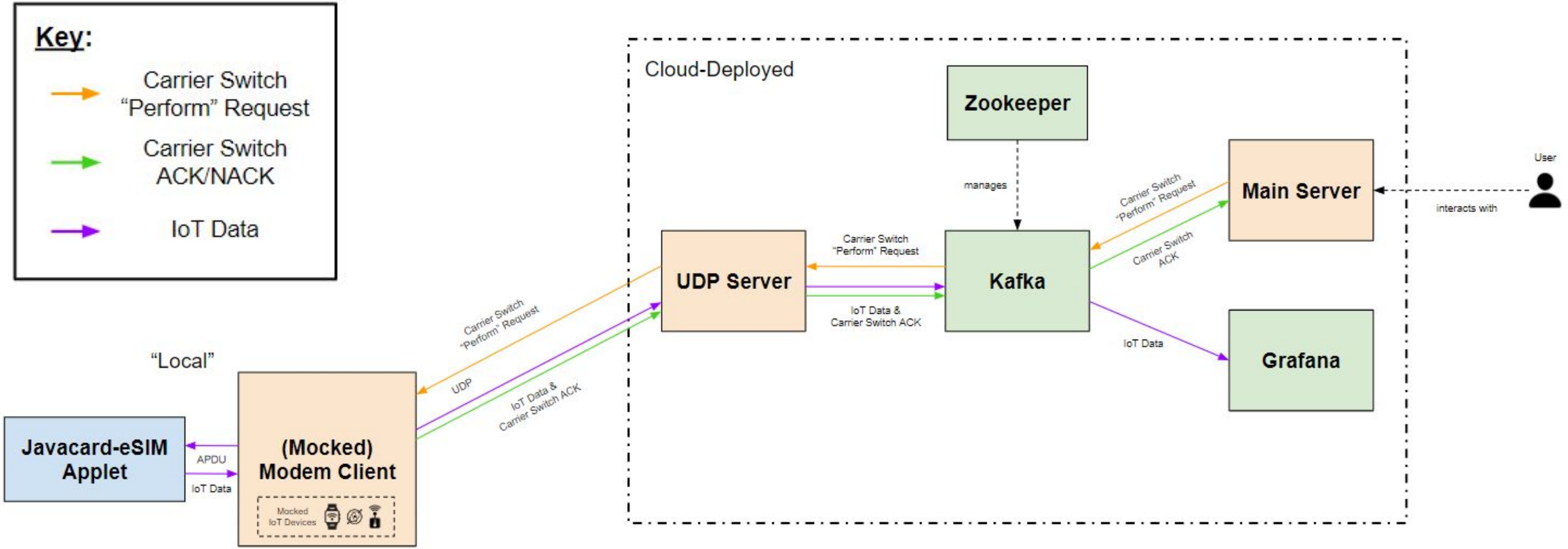
System Overview and Implementations

System Overview - Diagram



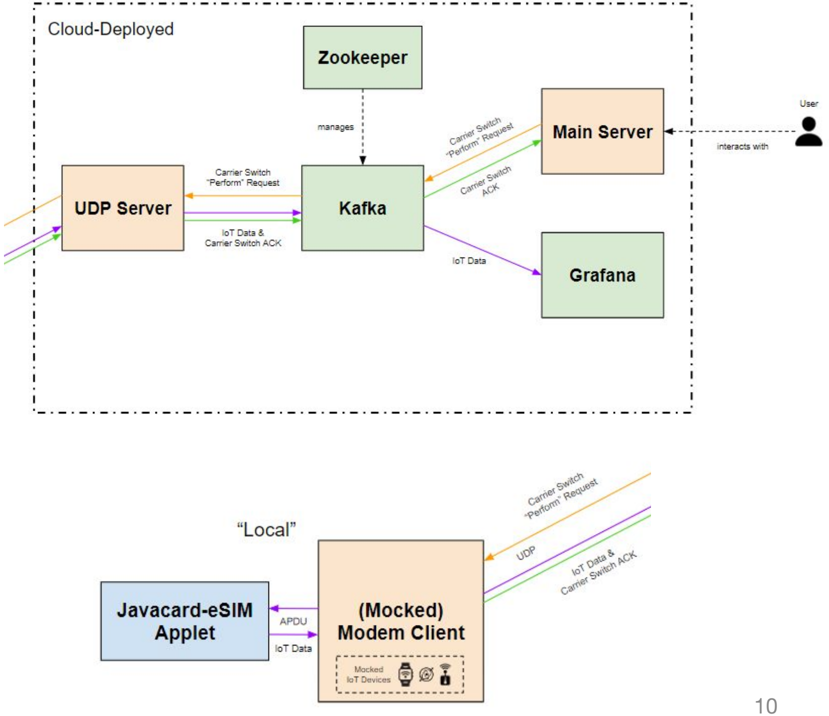
Mocked, for project scope

System Overview - Diagram (Real)



System Overview

- **Local:** Represents SIM-enabled 5G mobile client with IoT devices
  - **Python Modem Client:** mocks modem + IoT devices, speaks with SIM
  - **Javacard-eSIM Applet:** mocks SIM
- **Cloud:** Represents cloud server providing Carrier Switch functionality + IoT data visualization to user
  - **UDP Server:** Manages communication with modem
  - **Apache Kafka:** Distributed event streaming platform
  - **Zookeeper:** Manages Kafka
  - **Main Server:** Interface to user, exposing functionality + linking to Grafana
  - **Grafana:** Visualization of IoT data



SIM to Modem Communication

- APDU Commands
    - Traditionally work by modem sending request and SIM sending response
    - Proactive Commands: Using special APDU commands to give the SIM the ability to send requests to the modem/device
        - Works through FETCH command which will retrieve proactive command from SIM
            - OPEN CHANNEL: Tell modem to setup connection for future transmission of future data packets
            - SEND DATA: Tell modem SIM wants to send data
            - RECEIVE DATA: Tell modem SIM is ready to retrieve data
- Benefit of passing data through SIM
    - Can leverage the security capabilities on the SIM by using the home network keys to encrypt packets before sending out through channel (beyond project scope)
    - Less modification of modem/device code, since SIM handles creating/parsing the packets
- Applet modifications
    - On receive data command, parse to see if it is IOT data that needs to be passed back later, or IOT status packet that requires a “carrier switch”
    - Send back appropriate packet through SEND DATA

UDP Protocol Definitions

Flow and Topic Fields:

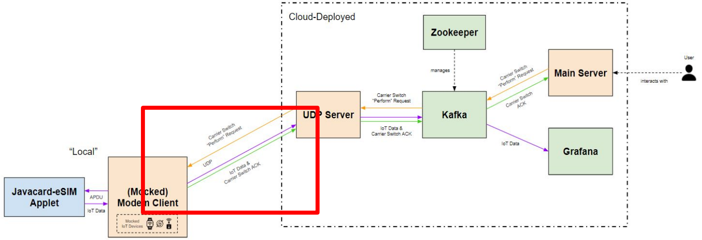
- **IoT**
  - **Data:** Holds actual IoT data
  - **Status (unused):** Send status information about IoT device
- **Carrier Switch**
  - **Perform:** Downstream request to perform a carrier switch
  - **ACK:** ACK/NACKing of actual carrier switch on SIM by Modem

General Packet Layout

Field	Size
Flow	4 bits
Topic	4 bits
Payload	(dependent on flow/topic)

Payload Summaries

Flow	Topic	Payload
IoT	Data	Device ID, Timestamp, Data Length, Data
	Status	Status Code
Carrier Switch	Perform	Request Carrier ID
	ACK	Status, Current Carrier ID



UDP Protocol Definitions (cont.)

Payload Field:

Flow: IoT
Topic: Data
Payload:

Field	Size
Device ID	4 bytes
Timestamp	8 bytes (microsecond precision)
Data Length	1 byte
Data	N/A (variable length)

Flow: Carrier Switch
Topic: Perform
Payload:

Field	Size
Carrier ID	4 bits
(padding)	4 bits

Carrier ID:

- AT&T = 0x0
- T-Mobile = 0x1
- Verizon = 0x2

Status:

- NOMINAL = 0x0
- IDLE = 0x1
- OFF-NOMINAL = 0x2

Flow: IoT
Topic: Status
Payload:

Field	Size
Status	4 bits
(padding)	4 bits

Flow: Carrier Switch
Topic: ACK
Payload:

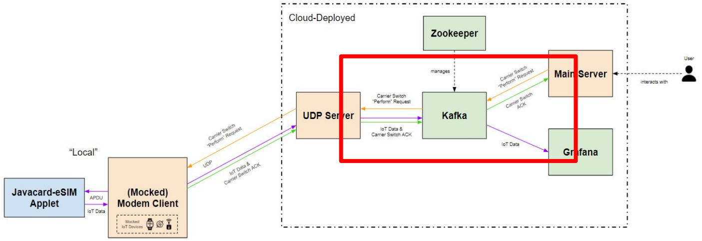
Field	Size
Status	4 bits
Carrier ID	4 bits

Status:

- ACK = 0x0
- NACK = 0x1

Kafka Topic Configuration

- **Downstream**
  - "downstream-request": Contains all downstream requests to Modem
- **Upstream**
  - **Carrier Switch ACK:**
    - "carrier-switch-ack": Contains ACK responses from downstream carrier switch requests
  - **IoT Data (device nickname):**
    - "imu": IMU data
    - "gyro": Gyroscope data
    - "temp": Temperature sensor data



Kafka Topic Configuration (cont.)

What: IoT Data
Direction: Upstream (to Flask Server)
Kafka:

- **Topic:** "<nickname>"
- **Data:** Python dictionary, represented as a JSON string encoded 'utf-8'

Key	Value (Type)	Notes
timestamp	ISO format timestamp, microsecond precision	Represents timestamp that data was collected from IoT sensor (microsecond precision) Produced by datetime.datetime.isoformat()
data	int	Data collected by sensor. For now, this is a SINGLE INTEGER. Can easily be extended to a dictionary of (key, value) pairs in real deployment.

What: Server Requests (ACKs)
Direction: Upstream (from UDP Server, to Flask Server)
Kafka:

- **Topic:** carrier-switch-ack
- **Data:** Python dictionary, represented as a JSON string encoded 'utf-8'
  -

Key	Value (Type)	Notes
status	str	Represents the status (ACK/NACK) of the carrier switch on the actual Modem/SIM client.
carrier	str	Represents the current carrier held by the Modem/SIM client (if the carrier switch was successful, then this will reflect what was sent in the original downstream "Perform" request).

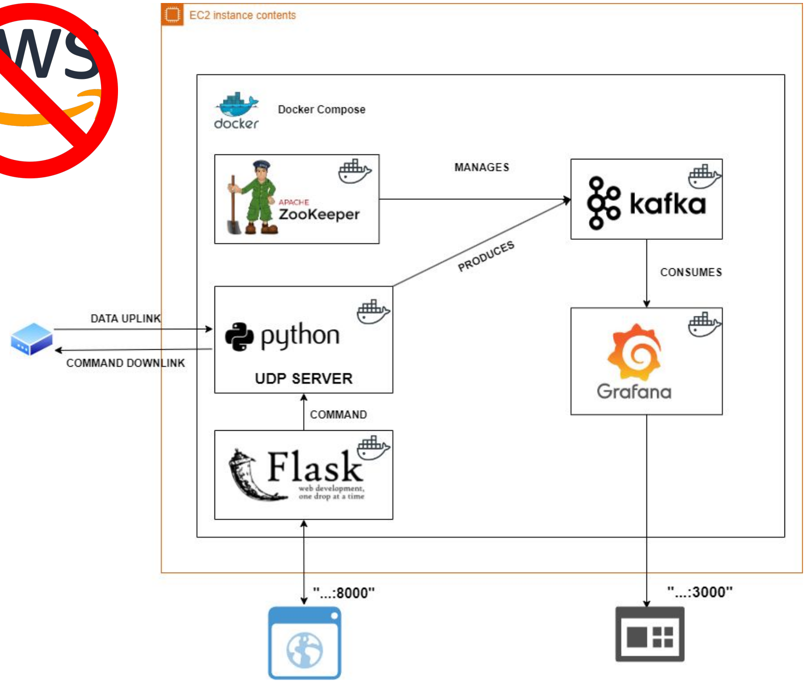
What: Server Requests
Direction: Downstream (from Flask Server, to UDP Server)
Kafka:

- **Topic:** downstream-request
- **Data:** Python dictionary, represented as a JSON string encoded 'utf-8'
  -

Key	Value (Type)	Notes
type	str	Represents the type of request that the Server has. Currently supported: - **"carrier-switch-perform"**: Perform a carrier switch by the Modem/SIM.
metadata	str	Represents the metadata used for the request. For now, we represent this purely as a string, and each message type can define this separately. In the future, perhaps this string represents a JSON of (key, value) pairs of data. **For carrier-switch-perform:** - String represents carrier to switch to (currently supported: "AT&T, T-Mobile, Verizon, Disconnect")

Cloud Deployment

- Docker Compose
  - Dockerized containers.
- Deployment - AWS.
  - EC2 Instance.
    - t2.large
  - Container vs. Managed Solution.
    - Vendor Lock-in.
- Next steps/future work.
  - Scale out strategy.



Implementation Challenges

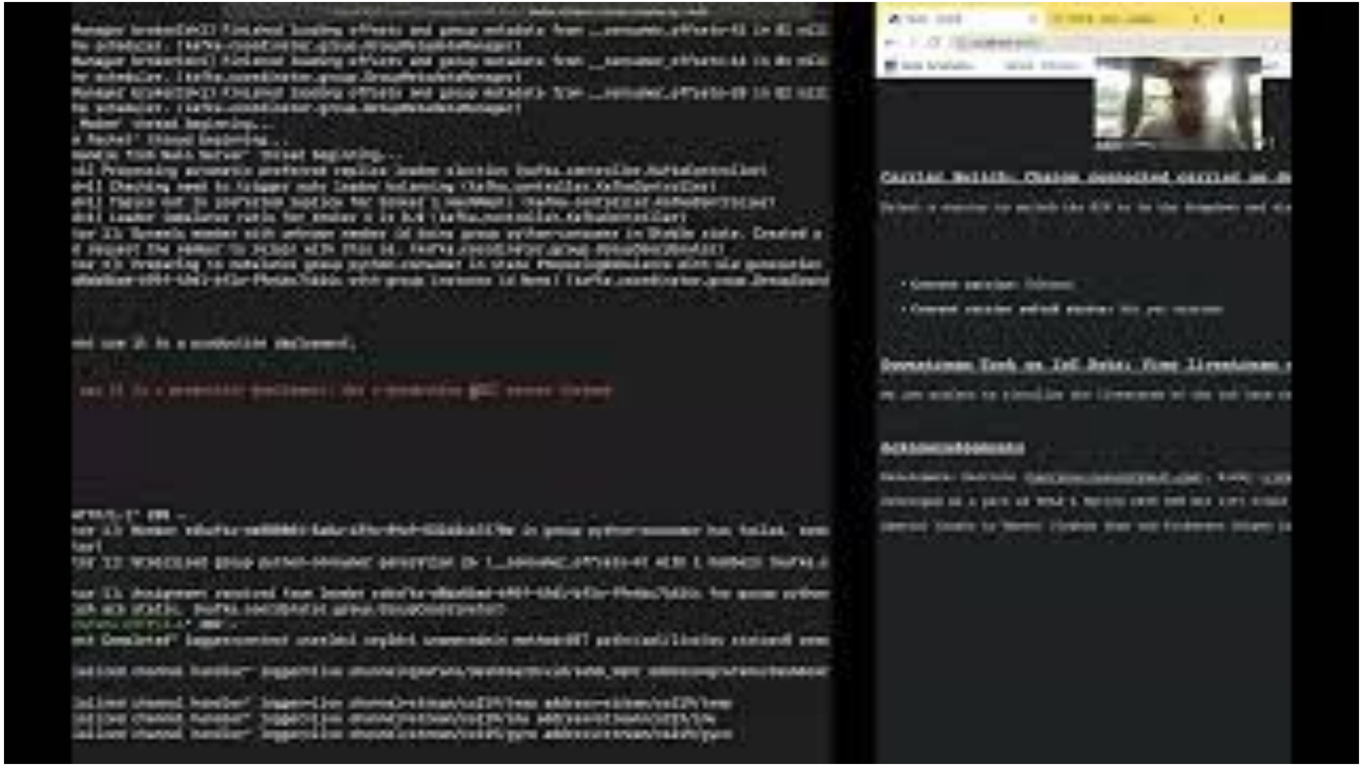
- Docker development environment issues
  - Known memory leak on Windows Docker Desktop
    - https://github.com/docker/for-win/issues/12944
- Thorough testing of subsystems needed other systems built out
  - Addressed via well-defined interfaces + building mocks of other systems, but lots of extra time spent
- CGNAT - UDP downlink communication.
  - Non-static modem IP address.
- Simplistic UI and flexible development on Flask vs interactive UI and complex development with ReactJS
  - Attempts on porting Flask frontend to ReactJS
  - Attempts on using ReactJS frontend with Flask as proxy

Current Status/Limitations

- **Full system implemented!**
  - Live cloud deployment
  - Locally-ran Modem Client able to connect send IoT data, receive Carrier Switch requests
- **Limitations (within project scope):**
  - Limited to one modem
    - Trivial to extend; nominal operation something to discuss/trade
  - Downstream tasks only involve visualizations
    - Ready for more, interact with Kafka API
  - 1Hz bottleneck on IoT data at the moment
    - Seemingly from the passing of IoT data through Kafka, however might just be due to misconfig

Demo

Demo Video



Insights Learned and Future Work

Insights Learned

- Apache Kafka **extremely flexible**
  - Fits distributed context very well
  - Great for **event streaming** and for tasks with **real-time requirements** (say, for live monitoring of critical systems) are better with Kafka
    - For large data and systems
- Docker-izing components make for **easy deployment**
  - On cloud or locally
  - Network bottleneck possible
- Interface definitions important to do **early**
  - Enables parallel work
  - Interface changes lead to issues if done too late

Future Work

- **Extensions to support:**
  - Multiple modem clients (trivial extension)
  - Exposure of more “mocked” metrics, such as Modem/SIM stats
  - Multiple users’ managing own suite of devices
    - Expose means of configuring/adding/deleting tracked Modems and/or IoT devices
- **Cleaner system:**
  - Unify servers to minimize slower over-network traffic
  - Further unify visualization with Main Server
  - Functional and interactive UI
- **Closer to a real-world scenario:**
  - Actual IoT Devices, non-mocked
  - Actual Modem with 5G multi-carrier SIM card
  - More interesting downstream tasks (Machine Learning..?)

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… **even after graduating**!

Code Repository

- Repository: https://github.com/harrisonCassar/CS219-Cloud-IoT-Management-SIM-Card
- Report soon completed, will be uploaded to the above repository.

Interact with our Site!

<u>Grafana Login Details</u>

Username: admin
Password: admin

Thank you!