Open-Source Prototyping of 5G Wireless Systems for UGV/UAV

DESIGN DOCUMENT

Team: 36
Client: Iowa State University
Advisor: Hongwei Zhang
Team Members: Nathan Whitcome, Ibro Tutic, Andrew Eschweiler, Samuel Stanek, William Byers, Nicholas Lorenz
Email: sdmay20-36@iastate.edu
Website: http://sdmay20-36.sd.ece.iastate.edu/

Revised: 4/25/2020

Executive Summary

Engineering Standards and Design Practices

- 3GPP
- E-UTRAN
- EURECOM
- IEEE
- Continuous Integration, Continuous Development (CI/CD)
- Coding Best-Practices

Summary of Requirements

Primary Requirement:

- Ensure per-packet communication reliability while achieving high throughput, low latency, and high reliability

Other Requirements:

- Utilize Open Air Interface (OAI)
- Utilize Simulation of Urban MObility (SUMO)

Applicable Courses from Iowa State University Curriculum

- CPRE 308 – Operating Systems
- CPRE 489 – Computer Networking and Data Transfer
- CPRE 430/530 – Network Security
- CPRE 543 – Wireless Network Architecture
- COMS 486 – Fundamental Concepts in Computer NEtworking

New Skills/Knowledge acquired that was not taught in courses

As a team during the first portion of the project, we studied wireless network scheduling algorithms. A particular emphasis was put on a few algorithms our advisor had published research on that focused on dynamic networks as well as increasing reliability.

From there we began familiarizing with the different software that the project required. Open Air Interface (OAI) is a massive codebase for simulating radio networks, and the knowledge required to use it had not been taught in any classes. Throughout the project members learned how to set up an operation environment for OAI, and how to run simulations on it. The other software not covered in classes is SUMO, which simulates vehicle positional data based on a map of an area. Members of the team had to learn how SUMO worked, and how to get the data from it that the simulations needed. For the interactions between OAI and SUMO, members gained experience in setting up and interacting with a TCP client software called TraCl.

In order to develop an improved scheduling algorithm, the team had to dig into several existing scheduling algorithms and analyze their functions. These algorithms were PRKS, CPS, and UCS. None of these algorithms were referenced in the courses our team had taken. An important skill was also developed here, as the algorithms had to be compared and analyzed for their strengths and weaknesses which is much different from the analysis of their time/space complexity.

Overall, most of the skills and knowledge gained by team members were due to the use of specific software and algorithms. Many of the general concepts taught in courses were applicable, but the specifics of this project allowed many of those same concepts to be deepened or varied.

Table of Contents

List of Definitions

Open Air Interface (OAI) – Open source software that simulates 3G, 4G, or 5G communication between devices.

Simulator for Urban Mobility (SUMO) – Open source software that simulates traffic patterns for a given part of the world.

Unmanned Ground Vehicle (UGV) – A mode of transport, such as a car, truck, or tractor, that is controlled remotely.

List of Figures

List of Tables

1 Introduction

1.1 Acknowledgement

This project would not be possible without the technical advice, planning advice, and material support of our faculty advisor Hongwei Zhang, his doctoral student Chen Ye Lim, and his graduate student Fancheng Wu.

1.2 Problem and Project Statement

This project deals with the lack of modern 5G implementations that allow for low latency and high throughput and reliability specifically in highly mobile networks. Current solutions for 5G networks do not guarantee any type of reliability between two nodes, especially in highly mobile environments like communication between ground or air vehicles. It involves developing and prototyping advanced 5G wireless solutions for unmanned ground and aerial vehicles, which have broad applications in domains such as connected autonomous transportation, smart agriculture, and advanced logistics. Furthermore, creating a highly reliable and low latency 5G network can allow self-driving cars to be safer and could potentially allow doctors to do surgeries from hundreds of miles away. This would keep people safer in more ways than one.

To create a solution to this problem, a new network scheduling algorithm needs to be developed that can help reorganize connections between nodes in highly mobile environments. Two proposed algorithms exist that can meet the requirements of this project, but they need to be implemented and tested in both simulation and real-world environments. These algorithms are called PKRS and CPS. PKRS is an algorithm that allows a network to achieve high reliability between two nodes that are close to each other. CPS takes this a step further and applies cyber physical scheduling (CPS) to extend PKRS to mobile networks, as it was originally intended for stationary nodes. CPS essentially extends PKRS to include data about the nodes positioning relative to each other without dedicating a lot of bandwidth solely to the transfer of positional information. This project will need to use a variation of the CPS algorithm in the MAC scheduling module of Open Air Interface to determine performance metrics associated with the new 5G implementation vs solutions that are currently available on the market.

1.3 Operational Environment

The operational environment for this project will be primarily in vehicles, which means the network will need to be able to operate under a variety of external conditions. However, this project does not deal with the hardware involved in the transfer of information, it is primarily the scheduling algorithm that allows the nodes to communicate with each other. However, we will need to ensure that things like storms, blizzards, and other natural disasters don't hinder the algorithms ability to reach high levels of packet reliability and throughput.

1.4 Requirements

Functional Requirements:

- High Throughput
- High Reliability
- Low Latency
- Interoperability with current solutions

Economic Requirements:

- Easy to simulate to avoide expensive hardware testing and implementations
- Code written to be maintainable

1.5 Intended Users and Uses

This project could have multiple types of end users, as it builds upon existing wireless network implementations:

- UGV's – Could be used in self-driving cars to determine traffic routes or vehicle actions. Smart agriculture could utilize self-driving tractors to automate farming tasks in the field.
- Surgeons – With low latency and high reliability, operations could be performed remotely, allowing better access to healthcare without a patient having to travel.
- Military Applications – Military vehicles could be operated remotely, while the low latency and high reliability still allow quick and effective responses to dynamic situations.

1.6 Assumptions and Limitations

Limitations:

- 5G signals have a relatively lower range and degrade quickly
- May not apply to UAV due to addition of verticality into positional data (2D to 3D)

Assumptions:

- Supported networks will not have major interference (i.e. mountains, jamming)
- Supported vehicle networks will not have a sparse amount of vehicles (nodes)

1.7 Expected End Product and Deliverables

Algorithm Simulation/Extension (May 2020)

This is the primary deliverable associated with the project. To achieve the requirements of the project, the algorithm needs to be implemented in Open Air Interface and tested by using SUMO to deliver traffic data and vehicle positions which Open Air Interface will use to simulate the network scheduling algorithm. To ensure that this deliverable meets the requirements listed above, the new scheduling algorithm will be tested and its performance measured to allow us to compare it to current solutions.

2. Specifications and Analysis

2.1 Proposed Approach

The proposed solution to develop a new 5G scheduling algorithm for highly mobile environments involves using Open Air Interface, an open source full stack network simulator and SUMO, a traffic simulator, to create a test bed for testing and developing a new algorithm. This approach involves configuring OAI to work in a multi-node environment to try and simulate large scale traffic simulations as well as possible. The network simulator follows the standards for wireless communication as directed by 3GPP and is continuously updated to include new features and releases from 3GPP wireless technology advancements.

The solution we propose needs have low latency, high throughput, and high reliability. To ensure these metrics are met, OAI and SUMO need to be integrated to provide a complete testing platform to ensure all our requirements are met. In our testing we used version 1.2.1 of OAI, which has been updated to 1.2.2 since we started this project. We previously used version .5.2, but that had a lot of bugs that caused our team a lot of problems and ultimately used up a lot of our time trying to fix. Version .5.2 had support for SUMO traffic data out of the box, but it was riddled with bugs that made it very difficult to set up, which was one of the reasons that we decided to go with the newer version of OAI.

We deployed the new version of OAI to a computer supplied by the ETG running Ubuntu 16.04 and configured a single eNB and single UE setup on the same machine. This proved to be difficult because OAI was designed to have separate components running on separate machines, so we had to find more documentation from other users that attempted something similar. This was required because we don't have the luxury of having many servers to run our tests with, so running many nodes on a single machine would be the most cost effective way to ensure that we can run our simulation without problems. This took a while as configuring the UEs and eNBs had many steps and missing one thing in a configuration file caused everything to seem like it was working, but further set up steps showed that the nodes could not communicate.

Going further, we had to implement an EPC (evolved packet core) to facilitate the connection between eNBs and UEs to the external network. The EPC handles transport of data from the UE to the external network, handles signaling related to mobility in the control plane, and contains a database of UE information that is used for session setup, user authentication and access authorization. Below is a diagram showing the design of a network that includes the EPC.

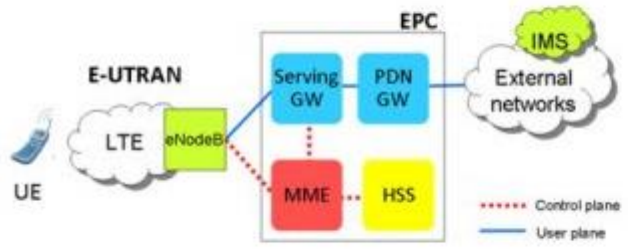


Figure 1: OAI Network Config

Configuring the EPC also was difficult because the setup guides that were written by OAI were for an older version of the EPC. This meant that we had to combine knowledge from many different sources and documents to figure out how to best handle configuring the connection between the EPC and the UE/eNBs. Once this was done, a connection between the eNB and the EPC could be established, leaving the registering of the UE information in the EPC HSS database for user authentication as the last step.

In terms of software, there has not been any progress on updating OAI to use the new algorithm as proposed by our advisor. The SUMO and OAI integration was a difficult task as it was hard to navigate the OAI codebase and find where in the network stack the location information from SUMO was needed. Also the problems with setting up the connection between the many components meant that testing on a real system wasn't possible until more recently, but the code to pull data from SUMO and update in near real time has been written and just needs to be integrated into OAI.

2.2 Design Analysis

So far we have configured the eNB and the UEs on a single server. The communication between the two is facilitated by nFAPI which uses a local loopback interface to allow the UE and the eNB to communicate with each other while running on the same server. This allows us to potentially scale the amount of UEs and eNBs we have to match whatever situation we would like to model. Unfortunetly, running these on the same machine does take a decent amount of processing power and there would be an upper limit to how many of these we could run on a single machine. This could be increased by getting more powerful servers, but there would still be an upper limit before we would need to investigate how to do a completely distributed network simulation, potentially in the cloud using a provider like CloudLab.

We also configured the EPC (evolved packet core) to allow for communication between the EPC and the eNB, which seemed to work as the EPC was able to report that there was one connected eNB. The only thing that the EPC is failing to report is the connected UE, which could be due to not having it registered in its local UE database. This was a problem that we were seeing a few times and even registering many times did not fix the issue. There were some bugs that other people reported with UE's connecting to the new version of the EPC, so we are not sure whether this issue that we are seeing is related at all to that.

We have also configured a Jenkins pipeline to manage continuous integration and continuous delivery on our fork of OAI. This allows us to make modifications to the code and check to see that all tests that were previously passing are still passing, to avoid breaking anything that was working before we started making changes. This was also set up on our system that runs OAI and seemed to work, although there were issues with many failing tests in the pipeline which came from lack of RAM and CPU on the server we were using. The CI/CD build system needed 12 GB of RAM on its own to run the virtual machines which facilitated the communication between many of the components of OAI for testing purposes and our machine had only 8GB of ram (a lot of which was taken up by running the UE and the eNB on the same system).

Our observations on our proposed approach is that OAI is a extremely large and complicated network simulator designed to be used in many, many different setups and configurations. The approach we are using, which involves using SUMO traffic data to update node locations inside of

OAI to allow the algorithm to schedule communication intervals between nodes was also something that proved to be difficult due to the sheer size and complexity of the OAI codebase. We have recently become a lot more familiar with the code base in trying to configure the network simulator for testing, so this portion will probably not be as difficult as it sounded at the beginning of the year.

In terms of the CI/CD build system, the ETG recently provided us with a better system to run the Jenkins server, but due to more important problems involving bugs with the communication between the EPC/UE/eNB, we had to put off moving over the old Jenkins system to the new server. This is something that would be important in a software heavy project, but we have just gotten to the point where we can begin implementing code inside of OAI.

The strengths of the proposed solution come from the running of the eNB and the UE on the same system and getting them successfully configured for communication. With them running on the same system, it means that we can use one system to simulate many eNBs and many UEs, which would allow the network simulation to scale much better rather than buying new systems to run one UE or one eNB everytime the network tester wanted to scale up the simulation. This did prove to be difficult to configure, as documentation for similar setups was missing critical steps and was outdated, meaning that we had to do a lot of digging for other configuration parameters that we might have missed.

Another strength of this solution is the build system. This would allow us to test our code against test that the OAI development team deemed essential to the running of the simulator. Without the build system, we could not be sure that previously working functionality was broken by any new changes we made to the codebase. All software projects should have some form of a build system to ensure old functionality isn't broken by new updates and it also allows maintainers of a repository to ensure that the master branch isn't broken by any code that gets merged into it that isn't properly tested. A screenshot of what the pipeline looks like can be seen below (the failing portion is because of memory limitations on our server, ideally with enough RAM this would not be a problem).

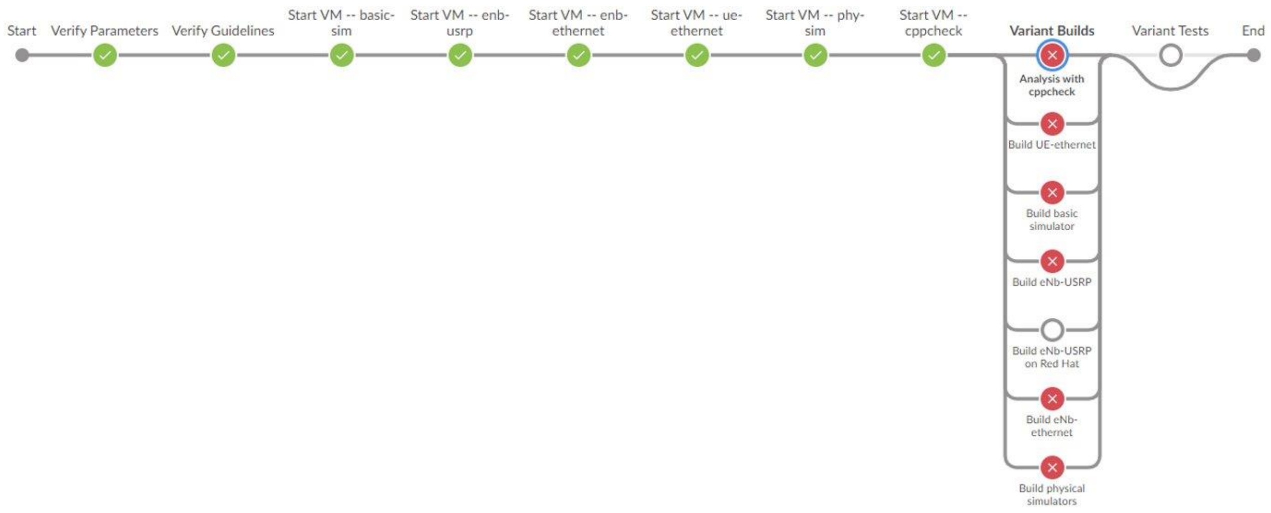


Figure 2: Jenkins Build System

A weakness of this setup is that it is very difficult to set up initially. Missing one small configuration step can still allow the component being configured to seem like it is running properly, but when moving on to the next piece that verifies the previously mis-configured components functionality, the lack of communication between the two components is evident and means that the developer has to backtrack to ensure that the other component is working before moving on to the next one.

Another weakness is the scaling of the system. There is an upper limit to how many UEs and eNBs can be ran on one system, so attempting to scale past that point would mean that more systems would need to be procured and configured. This opens up another potential issue in that OAI might not be configured for testing in this distributed manner and issues may arises from trying to use multiple UEs and eNBs on many different systems, although we did not get to the point where we attempted this configuration.

2.3 Development Process

The development process most applicable to our project was a variation of test-driven development. This is more applicable to purely software-based projects, but we found that we could extend some of the principles of TDD to our primarily systems-based project. When configuring the various parts of OAI (EPC, eNB, UE) we found it very helpful to configure each of the components, check each of the logs, ensure that there were no errors and go from there. This really helped us when configuring the four primary components of the EPC, the HSS database, the SPGW and the MMS. Each of these pieces needed to be configured, built, and ran independently and the log outputs from each one helped us understand if it was working properly or not. When running the final EPC after configuring each of the components, we did see some issues with the eNBs and UEs not connecting so we spent some time on the UE/eNB side of things trying to follow this same process and managed to find our problem within a week and this allowed us to fix it very quickly.

2.4 Conceptual Sketch

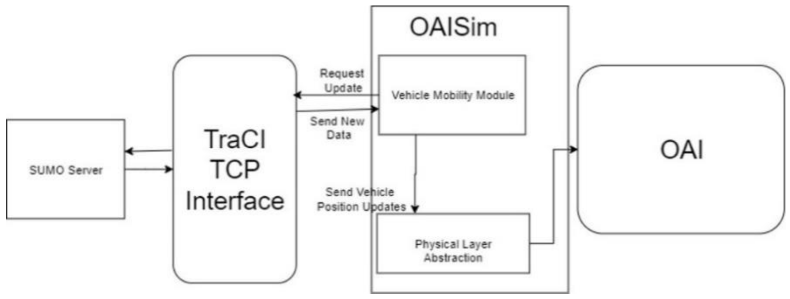


Figure 3: SUMO/OAI Simple Diagram

In the figure above, we have a SUMO server that runs a previously configured traffic scenario. This server has a TCP interface that allows it to communicate with other programs through the TraCI interface listed above. This interface includes an API that allows us to query for vehicle location data at a certain time and record the ID of the vehicle and its x and y position. This data is then fed into OAI through the Physical Layer Abstraction, which allows us to simulate interference between nodes due to their current positions. This positional data is then used by the MAC layer in OAI to determine the scheduling for communication between nodes that have a low chance of interfering with each other to ensure that their transmission isn't interrupted by other nodes attempting to communicate.

Figure 3 is a slightly simplified version of the interface between OAI and SUMO, ideally there would be an interface that could connect to either SUMO or a hardware based GPS system with little to no modification, but we did not get to that point in this project. That can be seen in Fig. 6 in Section 5.1.

The figure below shows the setup we are currently running in terms of the network configuration. The UE and the eNB are ran on the same system and are connected using the nFAPI over a local loop back IP interface. This allows us to run many nodes on a single machine.

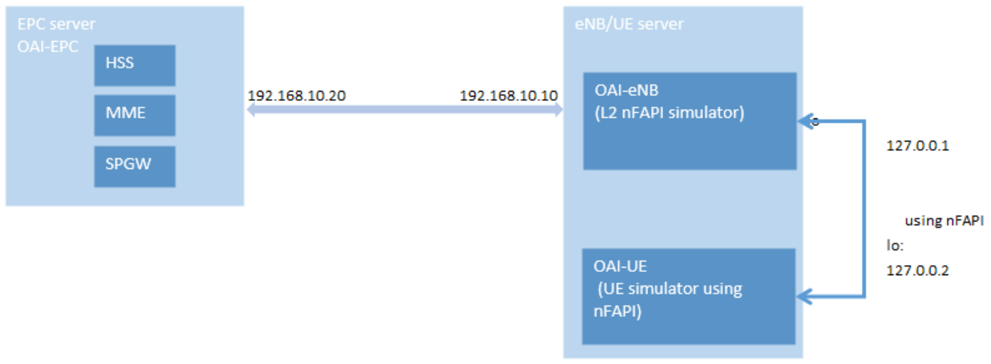


Figure 4: OAI nFAPI Network Config

3. Statement of Work

3.1 Previous Work And Literature

There are not similar products for mobile networks, current 5G implementations focus on nonmobile nodes (so like setting up a commercial 5G system for customers, like Verizon). This project focuses a lot more on situations like communications between air/land vehicles and how to optimize the network for maximum throughput and reliability. To begin researching the algorithm to use in this highly mobile network, the team familiarized itself with Dr. Hongwei's work on 5G scheduling algorithms, this research included the following published articles:

- "Scheduling with Predictable Link Reliability for Wireless Networked Control" [4]
- "Cyber-Physical Scheduling for Predictable Reliability of Inter-Vehicle Communications" [1]
- "Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control" [5]

These papers described initial versions of the algorithm that we are going to implement (in bold above) and how the test bed was set up to ensure the proposed algorithm performed as good or better than previous solutions. The bold paper Scheduling with Predictable Link Reliability for Wireless Networked Control (PKRS) defines the algorithm to achieve a high rate of reliability, throughput, and low latency. This algorithm works by defined an exclusion region around nodes to avoid interference in wireless transmissions through a specific parameter that depends on desired link reliability. (See figure below).

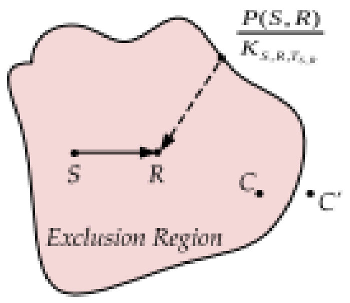


Figure 5: PRK Interference Model

The PRKS model defines and exclusion region for each link in the network (S, R in the Fig 1) around the receiving link (R in this case). The node C is in the exclusion region because the strength of the signal from C to R is greater than the ratio of the strength of the signal from S to R to the parameter K needed to take into account the presence of background noise and interference in the entire network. This parameter is chosen to maintain some minimum link reliability between two nodes.

The paper on Cyber-Physical Scheduling (CPS) for Predictable Reliability of Inter-Vehicle Communications applies the PRKS interference model to vehicular networks by extending the model to provide a geometric approximation to allow it to work in highly mobile networks. The initial PRKS model applies only to very low mobility stations. The CPS scheduling extends this model by instantiating the parameter K (same K as above) at every node. This algorithm leverages control theory to allow every link instantiated with the PRK model and the local signal maps that contain average signal power between S, R, and every other close-by node C that may interfere with the reference node. In this manner, the CPS algorithm is extremely similar to the PRKS model, except that is has been extended to account for highly mobile networks by having each node instantiate its own K parameter and have its own exclusion region for which it is responsible for.

3.2 Technology Considerations

There are two routinely used network simulators for this type of work: Open Air Interface (OAI) and NS-3. OAI is a much lower level simulator and is written in C, meaning that it is extremely efficient and mimics a real world setup better. NS-3 is more 'modern' simulator written in C++ and Python. It is easier to use and setup than OAI, but it offers a lot less flexibility. The biggest difference between NS-3 and OAI is that OAI can be ran on wireless communication hardware, which allows developers to test new 5G features in hyper-realistic environments. We choose OAI for that reason, because once we were finished with simulator the network purely in the software, we or other teams would have the ability to push the testing further into the real world by deploying to actual wireless communication equipment to verify results in a real world setup. NS-3 does not offer this level of simulation, but it is easier to simulate in software using NS-3.

A drawback of OAI is that is has extensive hardware requirements that make it difficult to set up. It requires an Intel CPU with all power saving features disabled to avoid any variance in clock speed that might cause issues when encoding and decoding packets being sent or received. We also needed to disable a feature called c-states in the BIOS, which allowed the CPU to enter a power saving mode which could affect the performance of OAI. Fortunately, the ETG was able to provide us a system with an Intel CPU and we were able to configure everything through the BIOS.

OAI also has extensive software requirements. We spent a lot of time working with the old version of OAI that had built in support for network simulations using SUMO traffic data, and that required Ubuntu 14.04 with a low latency kernel. After we configured this older version and spent a few months testing on it, we had to upgrade to a more recent and stable version of OAI, which required Ubuntu 16.04 and the low latency kernel that goes along with that version of Ubuntu.

The final piece of technology that we used is called SUMO. As mentioned previously, SUMO allows a user to configure a traffic simulation and is completely customizable. There are many different options that allow a user to configure traffic signals, roads, buildings, and just about any other possible traffic situation. Once this configuration is done, a simulation can be ran and through the help of an API provided by SUMO, the traffic data can be extracted from SUMO and used in other programs.

The older version of OAI that we were using (that had a good amount of issues) contained code that could be used to integrate SUMO and OAI. This code is no longer functioning in the new version of OAI and is almost 6 years old at this point, meaning that it might not even be compatible with newer version of the codebase. It does, however, give us a good start into seeing how to integrate the two software systems and get the traffic data to use for the algorithm we are developing.

3.3 Task Decomposition

Below is an outline of the various tasks for our team and their timeline. Each week contains key tasks for the specific teams to work on. Our team is currently broken up into those who work on the simulator and its integration with SUMO and those who do the algorithm analysis and implementation in OAI.

1. December
    1. Week 1
        1. Prepare for 30 minute design presentation
        2. **Continue working on physical layer porting.**
    2. Week 2
        1. Reevaluate roles on team, as initial development phase may be over and the same roles might not apply.
        2. **Continue working on physical layer porting.**
2. November, December, January: Algorithm Implementation
    1. Roles
        1. Product Owner - Figure out what features/work to prioritize
        2. Developer - Develop and implement the scheduling algorithm
        3. QA - Quality Analyst, ensure any new code is tested and verifies nothing old is broken through regression testing.
    2. Extending existing algorithms for more mobile application (UAVs/UGVs) [2-3 weeks]
    3. Continuous/iterative/spiral development, prototyping & testing [4-5 weeks
    4. Performance evaluation [3 weeks]
        1. Comparison vs current 5G implementations
3. February, March, April: nFAPI Simulation and network configuration
    1. Roles
        1. Algorithm developers – Work on porting algorithm to OAI
        2. SUMO/OAI Integrators – Work on integrating the new version of OAI with SUMO
        3. Network Configurators – Work on configuring the network for simulation
    2. Tasks
        1. Configure UE and eNB on single system
            1. Ensure proper configuration in config files
        2. Configure EPC on separate system
            1. MME setup
            2. HSS Database setup
            3. SPGW setup
        3. nFAPI used to allow UE and eNB to communicate on same system
        4. Continue work on algorithm to allow for V2x communication
            1. Highly mobile environment, vehicle to vehicle or vehicle to anything
        5. Integrate SUMO and OAI to allow for traffic data to be used inside of MAC layer in OAI

3.4 Possible Risks And Risk Management

There are a lot of risks in this project and we faced many issues when attempting to complete the final portion. The biggest risk in this project is the lack of knowledge for complex network simulators like OAI. To understand what the code is doing, knowledge of LTE and 5G networks is required to even begin understanding what is going on inside of the codebase. The only mitigation to this risk it to try and learn as much as possible about wireless communication, which is a difficult task because it is a very large field with many different technologies. It isn't feasible to learn everything about LTE and 5G communication, as it would take far too long and require a lot of PhD level experience from some of the resources we found to try and learn more about wireless communication. The best that we could do is learn about the specific parts that are applicable to our project (i.e. the physical layer and the MAC layer) and try to understand smaller portions of modules that occur near the modules that we will be working in.

Another possible risk is the bugs and debugging needed when trying to do a custom configuration like we are doing. There is documentation that shows how to set up a similar configuration, but it is pretty outdated, so it is difficult to understand all of the configuration files and what parameters are needed to successfully set up the system for network simulation. However, there is a mailing list where users can ask each other for assistance, so it will need to be leveraged in the case that we run into a large problem that we cannot fix.

OAI also requires very specific hardware requirements as mentioned in Section 3.2 so it would be extremely difficult to track down an issue that could be due to code or to an issue with the hardware that it is being ran on. OAI also has 13 total test cases that only check for things like segmentation faults and execution errors in very specific locations in the code. These tests would not give any indication that a scheduling algorithm is not work or where to begin debugging. This will make development much harder as it will be very difficult to verify that the code is doing what it needs to be doing. This can be managed by using a build system as we have done, but there still could be underlying problems that we are not able to see until a full-scale test of the system is done.

3.5 Project Proposed Milestones and Evaluation Criteria

The first key milestone in our project is getting OAI working. This is the first major portion of our project as our whole project relies upon the simulator working and producing accurate results. OAI is divided up into three subsystems, each which need to be compiled and ran separately. These subsystems mimic systems found in wireless networks and allow for maximum modularity and customization. Going further, to simulate a full network there are three critical pieces:

- UE (like a cell phone)
- eNB (radio tower)
- EPC (evolved packet core)

The UE and eNB will be ran on the same machine to save money to avoid having to get many machines to scale to many nodes. The EPC must run on its own machine and acts as a gateway from the 5G network to the external internet. It also does some scheduling, handle user authentication, voice/session creation, and handles some signaling in the control plane. A successful network configuration means that all these pieces will be able to connect and communicate with each other. To test this, the EPC outputs configured and connected components, so to ensure that communication between all the components is possible we just need to check the EPC logs and make sure that it is properly connected to the UEs and eNBs.

The second key milestone will be getting the OAI and SUMO integration working and figured out. This will be another integral part of our project because we will need to use the traffic data to validate that our algorithm produces the results that we expect. This is a difficult portion because finding where in OAI the traffic code needs to be is difficult considering the size of the code base and complexity. To test this, we can create a simple traffic scenario and see if the nodes in OAI are having their positional information updated by printing data to the screen every time a node updates its position.

The third milestone is implementing the algorithm in OAI and checking its validity. This can happen outside of SUMO integration just to make sure that everything builds properly, but to test this milestone we will need SUMO and OAI to be completely integrated to ensure that the algorithm is functioning as expected and that the final milestone below is met.

The final milestone will be seeing if the algorithm that we create meets the required project specifications:

- Low Latency
- High Throughput
- High Reliability
- Interoperability with current solutions

For the final milestone, we need to be able to test with the network simulator and SUMO to ensure that the nodes are having their positions updated inside of the network simulator and that the packets being transmitted between all of the pieces of OAI are being correctly scheduled by the scheduling algorithm that we implement.

3.6 Project Tracking Procedures

To track our project our group primarily uses Trello to track who is working on what and to assign tasks to people. We have five different columns: To-Do, Doing, Testing, and Accepted to manage the work. There are a few other columns that are primarily for organizing Status Reports/Project Information and other bookkeeping items. When new features are created, they are put in the To-Do column and people can move them forward as they start working on them and complete their work. Once in the Testing phase, another person tests the feature according to the criteria that is was designed for and then moves to accepted once the feature is completed. Additional tracking procedures are handled through our Microsoft Teams channel and we keep a lot of relevant files there.

3.7 Expected Results and Validation

The desired outcome of this project is to provide a complete network simulator that is integrated with SUMO and implements a new scheduling algorithm that allows it to achieve high reliability, low latency, and high throughput. To verify this works at a high level, we will need to run the simulation with the new algorithm and allow it to retrieve data from SUMO. From here we can perform measurements at the physical layer to see the current network statistics and compare to other algorithms to verify that there is improvement for these metrics.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

Assignment/Task	9/9-9/23	9/23-10/7	10/7-10/21	10/21-11/4	11/4-11/18	11/18-12/2	12/2-12/16	Winter Break	1/13-1/27	1/27-2/10	2/10-2/24	2/24-3/9	3/9-2/25	3/25-5/1
Read 4G/LTE Book	🟩	🟩	🟩											
Research PRKS Paper	🟩	🟩	🟩											
Research CPS Paper	🟩	🟩	🟩											
Research SUMO	🟩	🟩	🟩											
Research OAI and specific Install Steps				🟩	🟩	🟩	🟩							
Set Up Hardware/Operating Environment				🟩	🟩	🟩	🟩							
Install OAI					🟩		🟩							
Install SUMO				🟩	🟩	🟩								
Verify/Run OAI on Server							🟩		🟩	🟩	🟩	🟩	🟩	
OAI/SUMO Integration							🟨		🟨	🟨	🟨	🟨	🟨	
Algorithm Development						🟨	🟨		🟨	🟨	🟨	🟨	🟨	
Verify and Test Simulation														🟥
Writing Report												🟩	🟩	🟩
Compare Simulation Results to Control														🟥
Hardware Deployment														🟥
Finalize Report and Project														🟩

Table 1: Project Schedule Gantt Chart

This timeline has three main "stages" of the project. The first stage, from 9/9 to 10/21, is the research stage. For this stage, the team will be researching 4G, LTE, and 5G radios, wireless networking algorithms, and the open-source software we will be using. This is extremely important because our team does not have much background in these areas, and knowledge with these topics is necessary for success. We also will not have some of the required resources to progress meaningfully on our technical tasks until late October. This stage has no deliverables.

The second stage, from 10/21 to 2/24, is our technical stage. During this stage, the team will begin setting up the hardware and operating environment, installing OAI and SUMO for use in the third stage. The bulk of our technical work is contained in the OAI/SUMO integration and algorithm development. Our goal is to finish these tasks before winter break so that we have plenty of time to test and gather data for the reports. There will be no deliverables to the customer from these tasks, but our internal deliverables will be a working interface between OAI and SUMO, and a working eNB MAC layer in OAI for the scheduling algorithm.

The final stage of the project, from 2/24 to 5/1, is the verification and documentation stage. We will be working to test our implementations from the technical stage and produce data that we can use in our reports. The reports are the main deliverable, and hardware deployment is a secondary deliverable at the end of the spring semester. Unfortunately due to our technical stage taking more time than anticipated, we will not be able to complete all of our tasks during this stage.

4.2 Feasibility Assessment

We believe the majority of the project is feasible, however the hardware deployment deliverable may be unfeasible. The main challenges we have identified lie with the OAI codebase. The existing codebase is going to be difficult to work with and we are expecting to spend a lot of time debugging and learning how parts other than what we're developing work. The existing interface between OAI and SUMO also comes with known pre-existing issues, which means effort will be required before we are even able to start our integration. We don't expect these issues to prevent the delivery of our comparison reports, but they will possibly be exacerbated when we switch from the hardware testbed to the hardware deployment. This could cause the deployment deliverable to be delayed too much to be finished by the end of the semester.

4.3 Personnel Effort Requirements

Assignment/Task	Estimated Hours (Total)
Read 4G LTE/5G Book	12
Research PRKS Paper	12
Research CPS Paper	12
Research SUMO	12
Research OAI and Specific Install Steps	12
Set Up Hardware/Operating Environment	5
Install OAI	5
Install SUMO	5
Verify/Run OAI on Server	15
OAI/SUMO Integration	30
Algorithm Development	45
Verify and Test Simulation	15
Writing Report	20
Compare Simulation Results to Control	10
Hardware Deployment	30
Finalize Report/Project	20

Table 2: Estimated Project Tasks and Hours

4.4 Other Resource Requirements

This project will require the use of hardware capable of running both OAI and SUMO. A test bed of hardware specifically set up for OAI will facilitate quick and accurate simulations, reducing the need for implementation specific hardware. For team related code repository management and software access, a Linux box will also be needed that can be connected to via SSH. This system will have OAI and SUMO installed so that basic changes and tests can be run without the test bed.

4.5 Financial Requirements

No financial requirements have been defined for the project. The software we are using such as OAI and SUMO are both open source. The faculty advisor has also indicated that any hardware needs for the project have been or will be taken care of at no cost to the project.

5. Testing and Implementation

5.1 Interface Specifications

The only interface that we are working on for this project is an interface between OAI and SUMO. This interface is particularly important to design well, as one end needs to interact with OAI and the other needs to be flexible enough to obtain positional data from either SUMO or a hardware-based GPS system. The figure below shows how this would interact between SUMO and OAI or a GPS system.

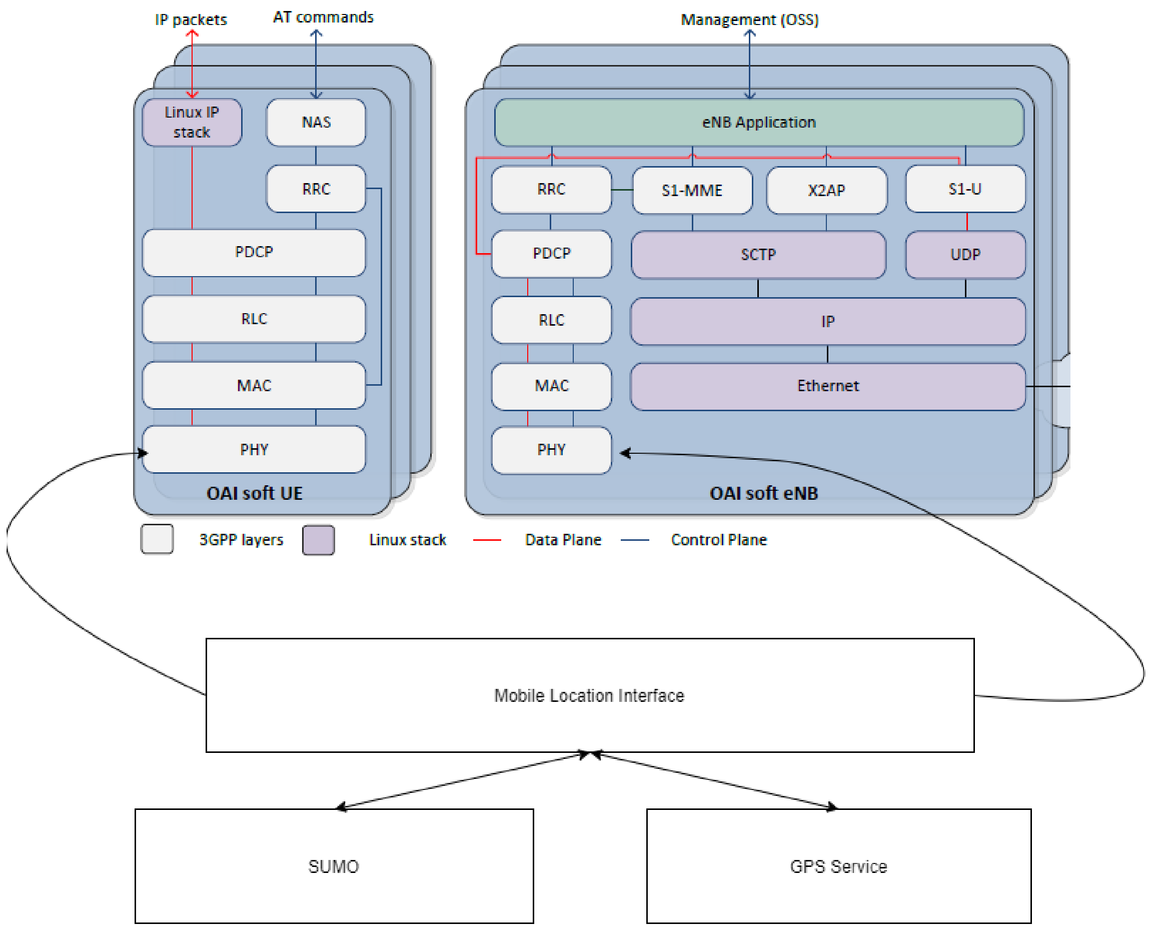


Figure 6: Mobile Location Interface

The mobile location interface would deliver location data to either the MAC or the PHYS layer. The MAC layer is where the packet scheduling is handled, so it would need the location information for the current node and nodes around it to determine what transmissions to schedule and when. The PHYS layer would be responsible for encoding and decoding incoming and outgoing packets so the current nodes location information would need to be encoded somewhere in the packet so other nodes have access to its location. The location information can be passed from layer to layer through the control frames, so either the MAC layer would directly communicate with the mobile location interface and pass the data to the PHYS layer for transport or the PHYS layer would handle accessing the data and pass the data to the MAC layer.

5.2 Hardware and software

The hardware used in this project is just a server provided by the ETG. This is needed to run a the UE and eNB side of a network simulator. The software we are using for testing is Open Air Interface and SUMO, along with a Jenkins build system to handle continuous integration/development and to help with regression testing.

Open Air Interface allows us to conduct purely software-based network simulations and allows us to eventually deploy the network to physical hardware. This feature of OAI gives us the ability to test the network in a purely software setup, which is low cost, and deploy to hardware once the software simulations show us positive results.

5.3 Functional Testing

The functional testing that we will perform on the network simulator primarily involves running simulations in our nFAPI setup and verifying measurements in the physical layer of OAI. There are various integration and unit tests that run in the build system to verify that old functionality is not broken. These tests perform vary basic functions that test the basic functionality of the various components of OAI (PHYS layer, eNB, UE) and ensures that connections between the components are still possible and that no code is broken.

Acceptance testing can be done slightly easier because similar code has been written for the NS-3 simulator, however for a slightly different variation of the algorithm. At the very least we can confirm that our implementation of the algorithm meets or exceeds the performance of the previous version written for the NS-3 simulator.

5.4 Non-Functional Testing

Performance and usability testing will be done using SUMO. SUMO will provide positional data to OAI and will allow us to simulate the movement of actual traffic patterns. This will help us maintain near real world simulation situations and will help us determine how well our scheduling algorithm works.

5.5 Process

The primary form of testing in this project is from the Jenkins build system. This system runs multiple virtual machines that deploy components of the OAI simulator. The testing phase then ensures that each of these components are working independently and together, this pipeline can be seen in Fig. 2 in Section 2.2.

Below is a conceptual flow diagram of our process when configuring the network and attempting to run tests.

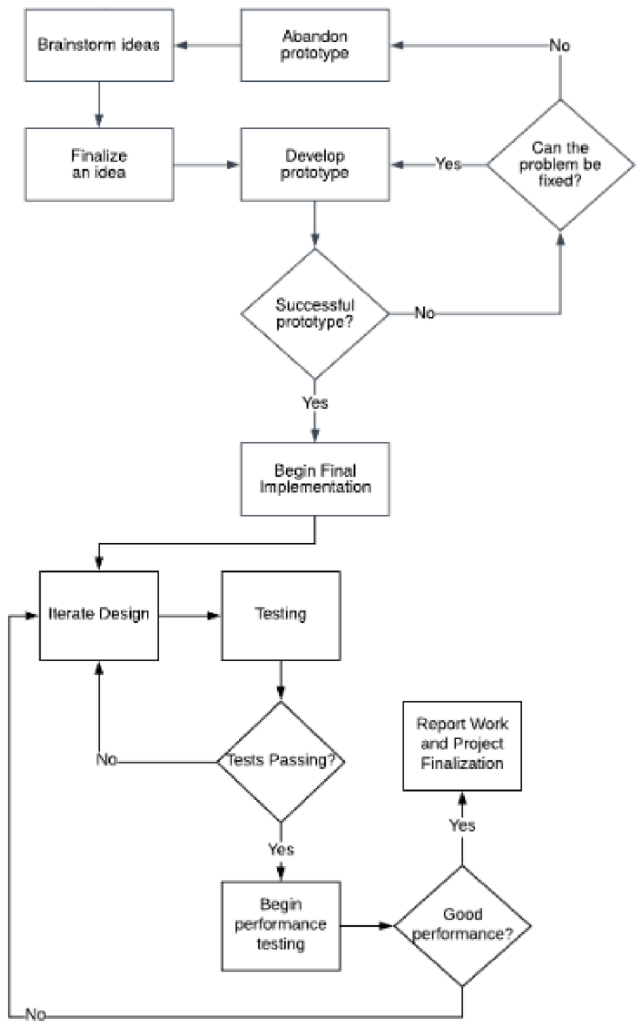


Figure 7: Process Flow Diagram

5.6 Results

Successes:

This project has been difficult, but we were able to get the OAI network almost fully configured and communicating. We built version 1.2.1 of OAI and used the nFAPI functionality to run a UE and an eNB on the same machine. We also successfully configured the EPC on a separate machine and successfully got the eNB and EPC communicating with each other. This part was very difficult as there was little documentation how to accomplish this task with the newer version of the EPC.

Another successful portion of the project is writing a module to obtain information from SUMO and process it for transport to another location. This module was written in C and uses the Python API provided by SUMO to obtain vehicle location information in the form of vehicle ID, time, x position and y position. This data can then be given to OAI to allow for the new scheduling algorithm to schedule the transmissions between nodes that have a very low probability of interference to allow for high reliability, low latency, and high throughput.

Failures:

Due to the complex nature of this project and the large amount of knowledge required to run the simulator, there have been a few failures. One of these is the lack of very much of our own code being put into the simulator to test the new algorithm. It took a lot of time to get the simulator up and running in a semi-usable state, so it wasn't possible to test any new code in a full integration of the system (outside of the build system). This made manually testing any new changes fairly difficult and ultimately led to us not having time to implement this portion of the project at the time of writing this report.

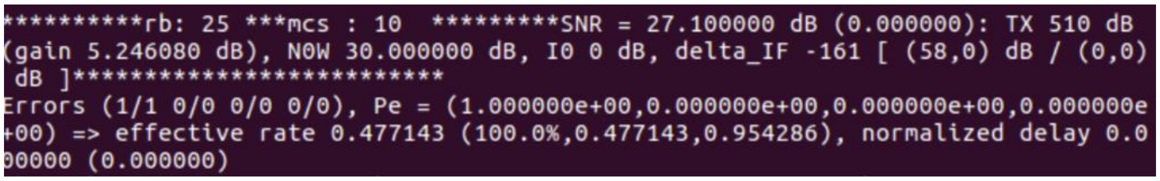
This is something that we are actively working on and hope to have more progress before the semester is over since the simulator is in a fairly good state. Another failure is that the lack of hardware resources for the build system meant that it wasn't fully functional and receiving those resources late while having a lot of other things to work on meant that nobody could dedicate time to trying to get it set up again.

Things We Learned:

We learned many different things about wireless communication, SUMO, and OAI through this project. Nobody on our team had very extensive knowledge with highly optimized low-level programming, complicated network stacks, and most importantly, knowledge about wireless communication and standards. Through this project we learned many things about the individual components that make up OAI and how wireless communication works at a high level. Through working with OAI and trying to understand the UE/eNB/EPC, we gained a lot of knowledge on the functions of each individual component and how to configure the network for simulation.

We also learned a lot about OAI at a higher level and how is has many different configuration options that allow it to be highly versatile. This is both a good thing and a bad thing; having a lot of configuration options means that it can be used in just about any situation, but this also opens the door to potential bugs and issues due to custom configurations that OAI might not be intended for.

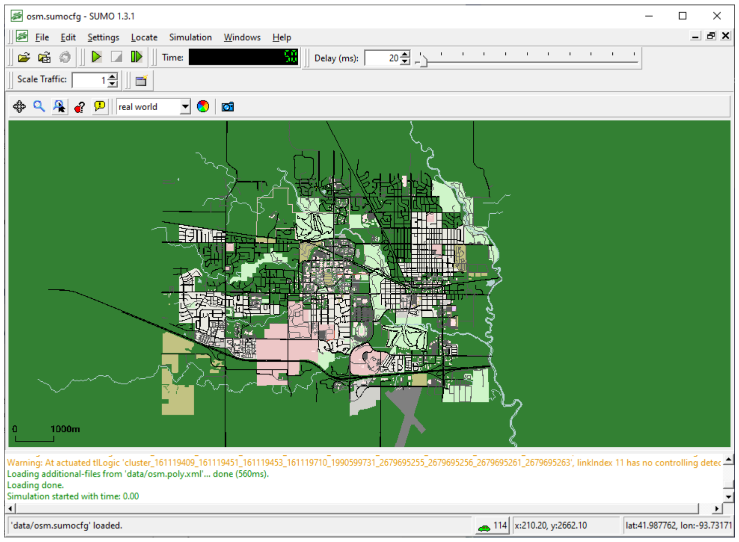
Modeling and Simulation:



```
**********rb: 25 ***mcs : 10  *********SNR = 27.100000 dB (0.000000): TX 510 dB
(gain 5.246080 dB), N0W 30.000000 dB, I0 0 dB, delta_IF -161 [ (58,0) dB / (0,0)
 dB ]***************************
Errors (1/1 0/0 0/0 0/0), Pe = (1.000000e+00,0.000000e+00,0.000000e+00,0.000000e
+00) => effective rate 0.477143 (100.0%,0.477143,0.954286), normalized delay 0.0
00000 (0.000000)
```

Figure 8: Upload Test

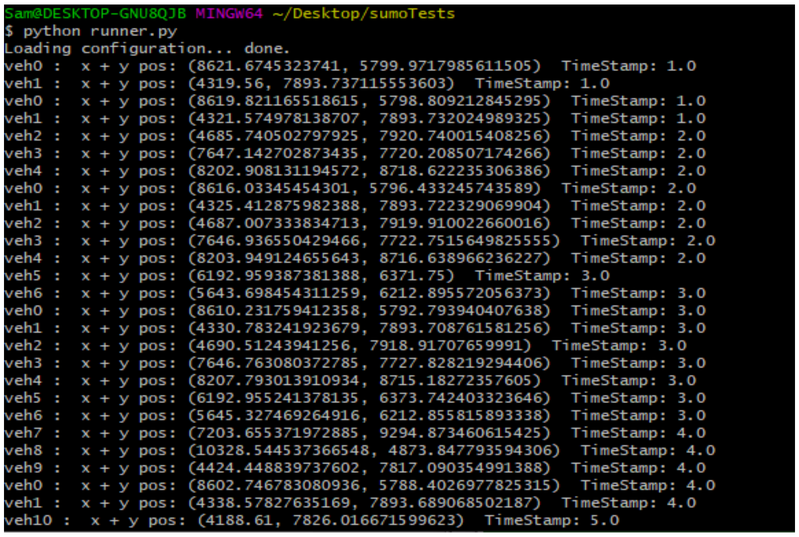
Figure 8 shows an upload test ran in OAI. This tests the connection between the UE and the eNB and checks various performance factors and error rates. Things like speed, signal strength, error rates, packet losses, and latencies are available from this data.



```
Warning: At actuated tlLogic 'cluster_161119409_161119451_161119453_161119710_1990599731_2679695255_2679695256_2679695261_2679695263', linkIndex 11 has no controlling detec
Loading additional-files from 'data/osm.poly.xml'... done (560ms).
Loading done.
Simulation started with time: 0.00
'data/osm.sumocfg' loaded.    114    x:210.20, y:2662.10    lat:41.987762, lon:-93.73171
```

Figure 9: ISU In SUMO

In Figure 9 above, we simulated the Iowa State area and designed a simulation where vehicles can move around on the streets that exist around campus. This helps us understand how to use SUMO and how to begin exporting vehicle positional data from SUMO into OAI for our simulations.



```
Sam@DESKTOP-GNU8QJB MINGW64 ~/Desktop/sumoTests
$ python runner.py
Loading configuration... done.
veh0 :  x + y pos: (8621.6745323741, 5799.9717985611505)  TimeStamp: 1.0
veh1 :  x + y pos: (4319.56, 7893.737115553603)  TimeStamp: 1.0
veh0 :  x + y pos: (8619.821165518615, 5798.809212845295)  TimeStamp: 1.0
veh1 :  x + y pos: (4321.574978138707, 7893.732024989325)  TimeStamp: 1.0
veh2 :  x + y pos: (4685.740502797925, 7920.740015408256)  TimeStamp: 2.0
veh3 :  x + y pos: (7647.142702873435, 7720.208507174266)  TimeStamp: 2.0
veh4 :  x + y pos: (8202.908131194572, 8718.622235306386)  TimeStamp: 2.0
veh0 :  x + y pos: (8616.03345454301, 5796.433245743589)  TimeStamp: 2.0
veh1 :  x + y pos: (4325.412875982388, 7893.722329069904)  TimeStamp: 2.0
veh2 :  x + y pos: (4687.007333834713, 7919.910022660016)  TimeStamp: 2.0
veh3 :  x + y pos: (7646.936550429466, 7722.7515649825555)  TimeStamp: 2.0
veh4 :  x + y pos: (8203.949124655643, 8716.638966236227)  TimeStamp: 2.0
veh5 :  x + y pos: (6192.959387381388, 6371.75)  TimeStamp: 3.0
veh6 :  x + y pos: (5643.698454311259, 6212.895572056373)  TimeStamp: 3.0
veh0 :  x + y pos: (8610.231759412358, 5792.793940407638)  TimeStamp: 3.0
veh1 :  x + y pos: (4330.783241923679, 7893.708761581256)  TimeStamp: 3.0
veh2 :  x + y pos: (4690.51243941256, 7918.91707659991)  TimeStamp: 3.0
veh3 :  x + y pos: (7646.763080372785, 7727.828219294406)  TimeStamp: 3.0
veh4 :  x + y pos: (8207.793013910934, 8715.18272357605)  TimeStamp: 3.0
veh5 :  x + y pos: (6192.955241378135, 6373.742403323646)  TimeStamp: 3.0
veh6 :  x + y pos: (5645.327469264916, 6212.855815893338)  TimeStamp: 3.0
veh7 :  x + y pos: (7203.655371972885, 9294.873460615425)  TimeStamp: 4.0
veh8 :  x + y pos: (10328.544537366548, 4873.847793594306)  TimeStamp: 4.0
veh9 :  x + y pos: (4424.448839737602, 7817.090354991388)  TimeStamp: 4.0
veh0 :  x + y pos: (8602.746783080936, 5788.4026977825315)  TimeStamp: 4.0
veh1 :  x + y pos: (4338.57827635169, 7893.689068502187)  TimeStamp: 4.0
veh10 :  x + y pos: (4188.61, 7826.016671599623)  TimeStamp: 5.0
```

Figure 10: Information feed from SUMO

Figure 10 shows a live feed of information generated by SUMO when the program is running. In this instance, SUMO is started as a “server” and sends the info to the python script which is running as the “client”. The python script uses the TraCI (Traffic Control Interface) libraries to send commands to SUMO using a TCP connection.

Implementation Issues and Challenges:

In both semester we faced many implementation issues and challenges. During the first semester we spent a lot of time working with a very old version of the simulator that had bugs that were not addressed because we needed to use the physical layer abstraction that was present in that version. This Physical Layer Abstraction would have made our project much easier because it already had integration with SUMO, but there were too many problems in trying to set up the simulator for simulation and editing the code since none of the developers on the mailing list were familiar with the code that was four+ years old. This set back our project a good amount in the first semester and we finally started working with a newer and better simulator in early December.

This simulator proved to be easier to use, but still had many different configuration options with little to no documentation on how to set up specific configuration files. There are many different documents that show how to set up many different configurations, so we spent a lot of time searching between many documents on figuring out the small issues we were encountering, specifically with the EPC. Another problem we had was the fact that we could not run the UE and the eNB on the same system after version .5.2 of OAI. This was something that we spent a few weeks trying to figure out and ultimately found a small guide on how to use a loopback interface on the local ethernet connection to allow a UE and eNB to be deployed on the same computer. This realization allowed us to configure the UE and the eNB on the same computer and then we moved to the EPC to try and get a full simulation working.

The EPC has three distinct portions, the HSS database which contains user sim information needed for authentication with the network, the SPGW, which is essentially a gateway between the internal and external networks, and the MME, which handles network access control, mobility management, radio and resource management, amount other functions. Each of these pieces needs to be configured independently and data from the eNB and the UE is needed to successfully get the EPC to communicate with them both. We spent a few weeks trying to figure out how to configure the EPC to communicate with the eNB and it ended up being a small piece of missing documentation where we needed to configure the country code, mobile tracking code, and network tracking code.

Another problem we had was that the server we were using for our build system did not have enough memory to run the build system and OAI. When a build was running, the server essentially become unresponsive due to how much processing power it took to compile all of the files in the new build and then the build would fail because there is not enough memory to allocate to all of the virtual machines that are needed to test the system. Another problem with our server was the fact that is was constantly failing and due to the COVID outbreak, we were not able to go into the lab to fix everything in a timely manner, we instead had to wait for the ETG to go in and restart our system so we could get access to it again. This cost us time as well, since the server had gone down a few times and there were network issues that left the server inaccessible over spring break until the ETG had time to fix it.

6. Closing Material

6.1 Conclusion

Overall, we have made a good amount of progress on this project, however, due to the extremely steep learning curve in dealing with OAI and learning how to configure its many components, progress was a little slower than we anticipated when we first started on this project. This coupled with the fact that we spent the first couple months learning the older version of OAI and then moving to a new version that was completely redesigned left us learning the new architecture essentially from scratch. We made lots of progress on getting a single UE and eNB configuration on a single system and we made progress on connected the eNB to the EPC.

On the SUMO side of things, the team managed to get a module that could request updates from the SUMO API and feed results to another program. This just needs to be integrated with OAI at the PHYS or MAC layer and the rest of the logic should be handled.

The new scheduling algorithm we worked on throughout the duration of the spring semester has gotten to the point of carrying out the preliminary design. This means it has not yet been implemented in OAI, but the process of pseudocoding and verifying the functions of it have begun. While this is not as far as we hoped to get, our goal is to finish preliminary design by the end of the semester.

Even though we did not get a full simulation report delivered to our advisor, the work the team has completed will allow future projects a much clearer base to start with, and they won't have to work through the same issues again. Since 5G is a very active area of research, we hope that the work done in this project will allow other projects to make more significant advancements in much less time.

6.2 References

[1] C. Li, H. Zhang, J. Rao, L. Y. Wang and G. Yin, "Cyber-Physical Scheduling for Predicatable Reliability of Inter-Vehicle Communications," *2018 IEEE/ACM Third International Conference on Iternet-of-Things Design and Implementation (IoTDI)*, Orlando, FL, 2018, pp. 267-272

[2] N. Nikaein, M. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research," *Sigcomm Computer Communication* Review, vol. 44, no. 5, Oct. 2014

[3] I. Latif, F. Kaltenberger, N. Nikaein, and R. Knopp, "Large scale system evaluations using PHY abstraction for LTE with OpenAirInterface," *6th MC & Scientific Meeting*, Malaga, Spain, 2013, pp. 24-30

[4] H. Zhang *et al.*, "Scheduling With Predictable Link Reliability for Wireless Networked Control," in *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 6135-6150, Sept. 2017.

[5] Y. Chen, H. Zhang, N. Fisher, L. Y. Wang and G. Yin, "Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2133-2145, May 2018.