Report

Data-Diode Driven Secure Intersection Data Acquisition for Enabling CAV Operations

An IoT Initiative for Implementing a Commercial Off-The-Shelf Data-Diode for Seamless Traffic Light Data Access on a Handheld Platform

Student: Manish Kumar Krishne Gowda
Email: mkrishne@purdue.edu
Advisor: Walton Fehr
Email: walton.fehr@dot.gov
Supervisor(s): James V. Krogmeier, Andrew D. Balmos
Email(s): jvk@purdue.edu, abalmos@purdue.edu

September 2022 - August 2023

Abstract

The primary objective of this project is to develop a comprehensive solution for the acquisition, processing, and dissemination of Signal Phase and Timing (SPaT) data from Traffic Signal Controller (TSC) cabinets to enhance the security and accessibility of timing information for drivers via their handheld devices. At the heart of this innovative and cost-effective approach lies the Data-Diode, a sophisticated device housing two interconnected microcontrollers operating through a simplex communication link. The Data-Diode acts as the bridge between the TSC cabinet and the user's handheld device, facilitating the extraction and processing of SPaT data.

The process begins with one microcontroller establishing a connection with the TSC's Ethernet port. Utilizing the Simple Network Management Protocol (SNMP) interface, it securely acquires the SPaT data. The obtained data undergoes Cyclic Redundancy Check (CRC) encoding to ensure integrity protection and is then transmitted through a unidirectional UART interface to the second microcontroller, thus maintaining isolation from external networks for heightened security. The second microcontroller interfaces with a 4G Cell Modem to transmit the processed SPaT data to the NATS open-source messaging system, incorporating a unique identifier specific to each TSC. Multiple such TSC-Diode systems are set up to gather data from various intersections. The collected data from all the systems is categorized based on their individual unique IDs and then routed through their respective channels within the NATS system facilitating efficient organization and streamlined access. A backend script is employed to acquire GPS coordinates from the user's handheld device, pinpointing the nearest traffic intersection. Leveraging this information, the script retrieves relevant SPaT data from the database associated with the corresponding TSC. The user is granted access to this data through a user-friendly web app on their handheld device. The app dynamically presents real-time information about the status of traffic lights and the precise timing of light transitions for the relevant lane at the identified intersection.

The implementation of this innovative research product is anticipated to result in a significant reduction in the overall costs associated with the acquisition, distribution, and processing of intersection data. Further, by seamlessly delivering accurate timing data to drivers' handheld devices, the system contributes to a safer and more focused driving experience by reducing driver distraction, often arising from the non-anticipation of traffic light changes. The real-time data on the user's handheld device enables drivers to make informed decisions well in advance, eliminating the need for abrupt braking or acceleration due to unforeseen light transitions. By utilizing the Data-Diode system and its associated components, the project empowers the drivers with advanced knowledge of traffic light transitions, enhancing road safety and optimizing traffic flow. The integration of this technology into everyday commuting experiences has the potential to transform urban mobility by providing drivers with the tools they need to make more informed, efficient, and safe driving decisions.

Keywords: Data Diode, STM32, Traffic Signal Controller, Ethernet, SNMP, User Datagram Protocol (UDP), CRC, SPaT data, Cell Modem, NATS, Postgres

Contents

1. Introduction

In today's rapidly evolving transportation landscape, the emergence of Connected and Automated Vehicles (CAVs) holds the potential to revolutionize road safety while simultaneously curbing pollution, energy consumption, and travel delays. These innovative vehicles have prompted extensive studies, both in the past and ongoing, to delve into their operations within various subsystems of the transportation network, encompassing dynamic freeway weaving segments and intricate urban intersections.

However, translating the promise of CAVs into actionable operational decisions necessitates a paradigm shift in the acquisition and dissemination of critical infrastructure data. While existing methods of data collection and distribution are well-established, they often prove to be resource-intensive and time-consuming, leading to inefficiencies in decision-making processes. A glaring example of this challenge is evident in Gwinnett County, Georgia, where equipping a mere 20 intersections with Dedicated Short-Range Communication (DSRC) equipment incurs an exorbitant cost of $309,000. Such staggering figures underscore the imperative for more economical approaches to obtain and distribute essential data.

Traditionally, the collection and distribution of Signal Phase and Timing (SPaT) and Intersection Map (MAP) data involved intricate arrangements that incorporated computing systems and short-range wireless equipment at each intersection. These systems, while effective, come with substantial setup costs and operational complexities, thus motivating the exploration of novel, more cost-effective alternatives.

Amidst these technological advancements, the significance of traffic intersections cannot be overstated. These junctures serve as linchpins in transportation networks, impacting the lives of countless individuals. Consequently, the imperative of security becomes paramount. Recognizing this, the US Department of Transportation adopts a meticulous approach to the integration of innovative concepts within existing infrastructure, emphasizing security as a non-negotiable tenet.

The central focus of the present project revolves around the exploration of innovative and cost-effective methodologies for collecting intersection data, strategically aligned with the evolving landscape of traffic operations in the imminent era of CAVs. The overarching objective of this research endeavor is to introduce a transformative strategy that renders intersection data more widely accessible and economically viable. This objective is meticulously pursued through the development and deployment of a cutting-edge, low-cost data diode solution.

This innovative approach centers on ensuring the secure retrieval of traffic controller timing data while effectively addressing the burgeoning demands of data acquisition within the context of CAVs. To achieve this, the project strategically harnesses the concept of unidirectional data movement—a core tenet that underpins the proposed data diode framework. By engineering data flows to move exclusively from the traffic infrastructure to mobile devices, a powerful security layer is inherently established. This unidirectional data movement ensures that sensitive information remains safeguarded against any attempts at external manipulation, thereby mitigating potential vulnerabilities and enhancing the overall integrity of the data acquisition process.

The device designed to facilitate such unidirectional data transfer is termed the Data-Diode. The implementation of this diode fundamentally transforms the acquisition landscape, as it empowers the flow of information in a singular direction—outward from the traffic infrastructure to mobile devices. This design not only prevents unauthorized access from external entities but also guarantees the preservation of data accuracy and security.

The pivotal contribution of the data diode lies in its ability to enable the seamless and secure transmission of traffic controller timing data to mobile devices, ensuring that drivers and passengers are equipped with real-time insights. By integrating this unidirectional movement mechanism, the project not only safeguards the integrity of the data but also upholds the security of the traffic controllers themselves, as potential avenues for external interference are effectively minimized. Thus, it not only bolsters the accessibility of high speed intersection data but also engenders an unparalleled level of security.

Central to the data acquisition and processing strategy of this project are the MAP message and the NTCIP (National Transportation Communications for ITS (Intelligent Transportation Systems) Protocol) 1202v2 Traffic Signal Controller Broadcast Messages (TSCBM). These components play a pivotal role in capturing and processing essential intersection data.

The MAP message serves as a comprehensive repository of critical information regarding an intersection's configuration and regulatory attributes. It encompasses intricate details such as lane-level road geometry, permissible turning maneuvers at stop lines, and other regulatory data specific to an intersection or segment of roadway. This multifaceted message structure effectively conveys diverse road geometries, with particular emphasis on its "intersections" structure, which is of paramount importance within this context.

In parallel, the NTCIP 1202v2 Traffic Signal Controller Broadcast Messages (TSCBM) assume a vital role in this data acquisition and processing endeavor. These messages, systematically broadcasted by Traffic Signal Controllers (TSCs), encapsulate the Signal Phase and Timing (SPaT) information, a cornerstone element for understanding the current and future movements orchestrated by one or more signal controllers. Beyond SPaT, these messages also furnish crucial insights into various lane or intersection regulatory matters that fluctuate with the time of day, such as designating a lane for longer time during peak traffic hours. Appendix A.1 provides a link to the standard NTCIP Object Definitions for Actuated Traffic Signal Controller (ASC) Units.

The combination of these messages facilitates a comprehensive and granular understanding of an intersection's dynamics. The MAP message, with its lane-level geometry details and turning maneuvers, provides a spatial blueprint of the intersection's layout and operational rules. On the other hand, the TSCBM, with its SPaT information and regulatory data, captures the temporal orchestration of traffic movements and regulatory shifts.

Together, these messages synergistically contribute to the project's overarching goal of enhancing intersection data accessibility and accuracy. By harnessing the power

of the MAP and TSCBM messages, the project seeks to empower drivers with real-time information about traffic light status, ensuring safer, more informed commuting experiences. This integration of complex data structures not only fosters improved decision-making for drivers but also sets the stage for optimized traffic management in the context of CAV systems.

As elucidated in [1], each signalized intersection constitutes fundamental components that form the bedrock of signal operations. These components include a controller, a cabinet, displays (or indications), and typically, detection mechanisms. This composition is visually depicted in Figure 1. To further illuminate the interplay between these signalized elements and the users of the system, Figure 2 illustrates the essential interactions.

The detectors, encompassing vehicle, pedestrian, or bicycle sensors, play a pivotal role by transmitting messages to the controller as users approach the intersection. This communication forms the backbone of the system's responsiveness. Subsequently, the controller harnesses the input received from these detectors to orchestrate changes in user displays—typically designed for vehicles and pedestrians. These alterations in displays are predicated on the signal timing parameters defined by practitioners, thereby adhering to established guidelines.

The controller's role in allocating time to different users hinges on an intricate interplay between detection mechanisms and signal timing parameters, often encapsulated in controller settings meticulously programmed by practitioners. This interdependence dictates the pace and synchronization of traffic flow, optimizing efficiency and safety within the intersection environment. Thus, the symbiotic interplay between components, detectors, and practitioner-defined settings culminates in the seamless management of traffic flow, optimizing efficiency and safety for all users.

Traffic cabinets situated in close proximity to intersections, housing the vital electronic controls necessary for traffic light transitions. These cabinets serve as repositories of crucial data, pivotal for informed decision-making and thorough analysis. However, the acquisition process must not come at the expense of compromising the cabinets' integrity. When Traffic Signal Controllers within these cabinets interface directly with microcontrollers that engage in broader communication with external entities—wherein data is both collected and processed—an inherent security risk emerges. This exposure could potentially render the Traffic Signal Controllers vulnerable to exploitation by malicious actors, endangering the real-world management of vehicle traffic and thereby posing severe threats.

For a more concrete visualization, Figure 3 offers a glimpse into a prototypical cabinet and its internal components. This illustration provides insight into the physical structure and arrangement of these critical elements that collectively govern the functionality and orchestration of a signalized intersection.

Data-Diode is a comprehensive exploration into the realm of secure and efficient extraction of high-speed SPaT data from such TSCs. Leveraging cost-effective components such as 32-bit microcontrollers, compatible 4G cell modems, and advanced 4G Ultra Wide Band Antennas, the project aims to establish a robust solution that maintains affordability while safeguarding data velocity and veracity. In response to this

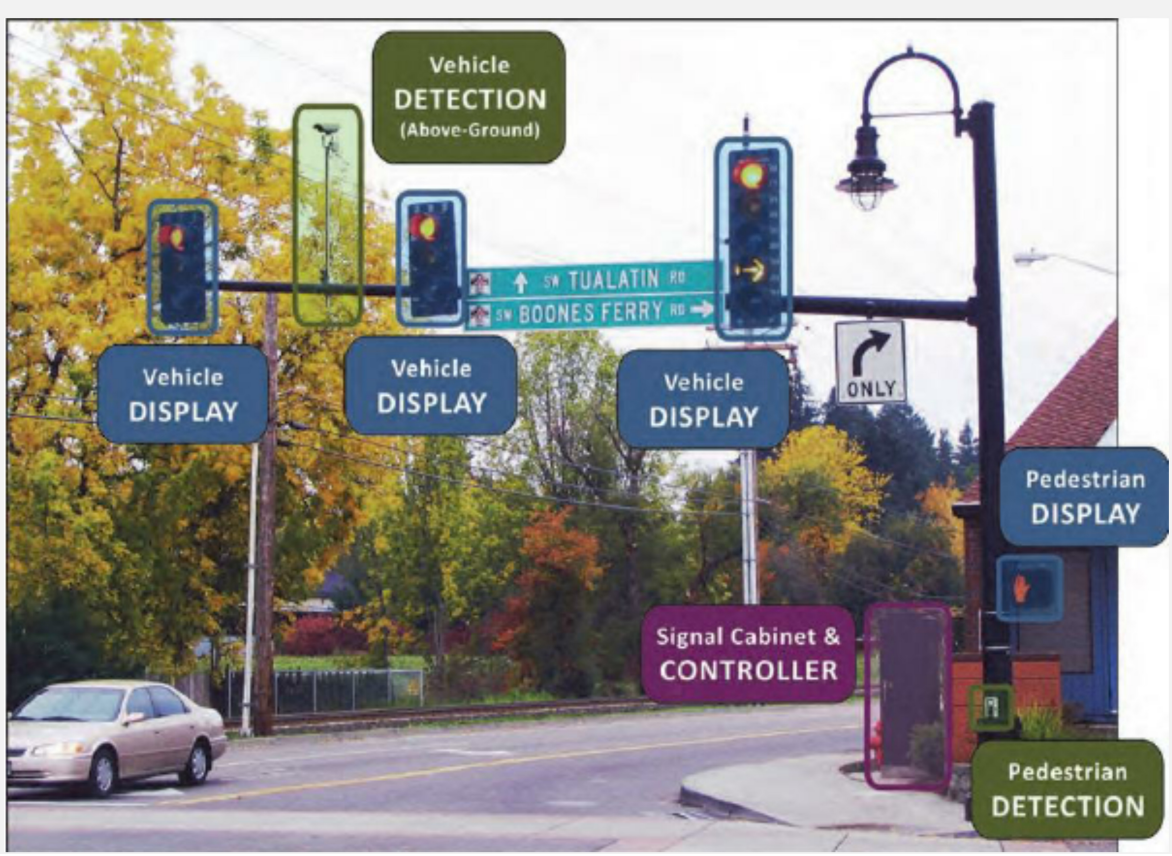

Figure 1: Typical Signalized Intersection Components

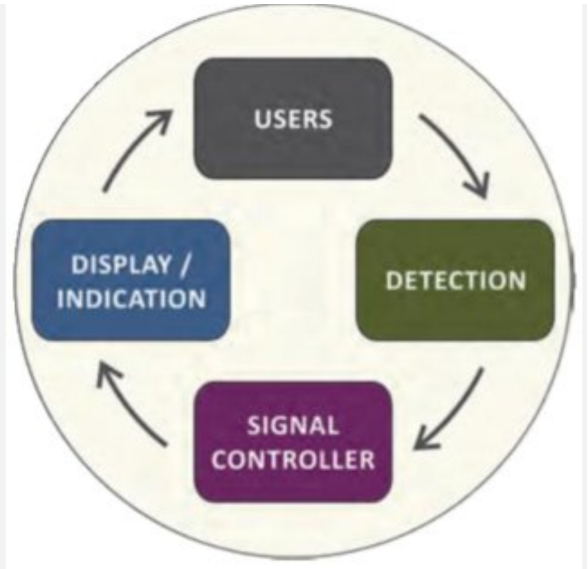

Figure 2: Generic Signalized Intersection Interaction

Figure 3: Typical TSC Cabinet

challenge, the proposal champions a two-microcontroller approach. The first microcontroller, housed within the cabinet, interfaces directly with the Traffic Signal Controller. Its primary role involves the extraction of intersection data, which is then transmitted through a single-directional simplex communication cable to a second microcontroller. This second microcontroller serves as the intermediary that relays the data to a remote server using the cell modem, subsequently enabling comprehensive analytics. The key innovation within this framework lies in the physical isolation of the first microcontroller from external communication networks. By severing any direct physical links between the microcontroller and external entities, the security of data extraction is fortified. This strategic isolation mitigates potential vulnerabilities, fostering a secure environment for data collection.

The SPaT data is wirelessly transmitted by the cell modem interfacing the second microcontroller to a NATS Server, where it is meticulously categorized based on sender identification embedded by the second microcontroller. NATS, known for its lightweight and open-source architecture, seamlessly supports distributed systems while adhering to the core tenets of performance, scalability, and ease of use. NATS simplifies the process of integrating additional TSC-diode units into the broader system. As new TSC-diode units are introduced, NATS effortlessly handles the integration of their data streams, ensuring smooth communication and coordination within the larger network. This streamlined scalability enables the system to grow in response to evolving traffic demands and changing intersection configurations without causing complications or disruptions.

Furthermore, the project encompasses a practical application that enhances user experience. Leveraging the location information derived from a driver's mobile device, the system identifies the nearest intersection utilizing stored MAP data. By accessing the corresponding SPaT information, the system promptly communicates the traffic light status to the user's handheld device. This real-time dissemination empowers users with crucial information, fostering safer and more efficient commuting experiences.

Later sections of this report provides a comprehensive breakdown of the project's intricacies, spanning from its constituent hardware and software elements to the meticulous workflow setup and operational methodologies. Through this multifaceted endeavor, the project endeavors to redefine the landscape of data acquisition and security within the realm of traffic operations, setting a precedent for efficient, secure, and user-centric transportation systems.

2. Methodology

The end-to-end architecture of the system comprises four distinct components, each tailored to specific functionalities:

1. Controller Side of the Data-Diode:
    - Programming the TSC to transmit SPaT data using SNMP over its Ethernet port
    - Programming an STM32 microcontroller to receive SNMP data from the TSC via the STM32's Ethernet port
    - Encoding data using a Base64 encoder, including the received SPaT data and a CRC value
    - Transmitting the encoded data over the Tx port of a simplex UART connection
2. World Side of the Data-Diode:
    - Programming an STM32 microcontroller to receive encoded SPaT data from the controller side of the STM32 through the Rx port of a simplex UART connection.
    - Interfacing the STM32 on the World side with a 4G LTE-compatible cell modem
    - Publishing the encoded SPaT data to a remote NATS server using the Modem Client, with the data destined to the NATS subject specific to "UNIQUE ID".
3. Server/Edge Side of the System:
    - Establishing a NATS server configured to continuously listen for data from all TSC-Diode systems
    - Decoding the base64-encoded data and parsing the fields of the NTCIP 1202v2 TSCBM SPaT data.
    - Associating the UNIQUE ID with the corresponding intersection and republishing the parsed data to the relevant intersection subject
    - Responding to queries from the app by providing information about the nearest intersection number, current traffic light status, and allowed maneuvers in the app's queried location.
4. Mobile App Side of the System:
    - Designing a user-friendly mobile application capable of capturing the user's GPS location
    - Querying the server for the nearest intersection number based on the user's GPS coordinates.

- Subscribing to the NATS topic specific to the intersection number to retrieve the parsed SPaT message.
- Calculating and displaying latency information through a dedicated script.

By breaking down the system into these discrete components, the complex processes of data acquisition, transmission, processing, and display are streamlined. Each component is specialized for its designated role, enabling efficient communication, parsing, and analysis of the intricate intersection data. This modular structure not only enhances the system's robustness and efficiency but also facilitates easy maintenance, scalability, and future enhancements.

The preceding sections have outlined the diverse components in conjunction with their intricate interconnections that collectively culminate in the realization of the Data Diode system. This comprehensive architectural structure is vividly depicted in Figure 4, providing a visual representation of the synergistic integration of these components. Furthermore, the practical manifestation of the system, as evidenced by a real-life demonstration, is vividly illustrated in Figure 5. Notably, the initial rendition of the data diode enclosure, Version 1.0, is meticulously showcased in Figure 5, offering tangible insight into the physical embodiment of this pioneering technology.

To further illuminate these intricate interconnections, interdependency features, as well as the operational intricacies of each individual component, a comprehensive exposition will be presented in Chapter 4. This chapter will delve into granular detail, elucidating the mechanisms through which each element collaborates within the system framework. Through this comprehensive exploration, readers will attain a profound understanding of the seamless orchestration that underpins the Data Diode system's functionality.

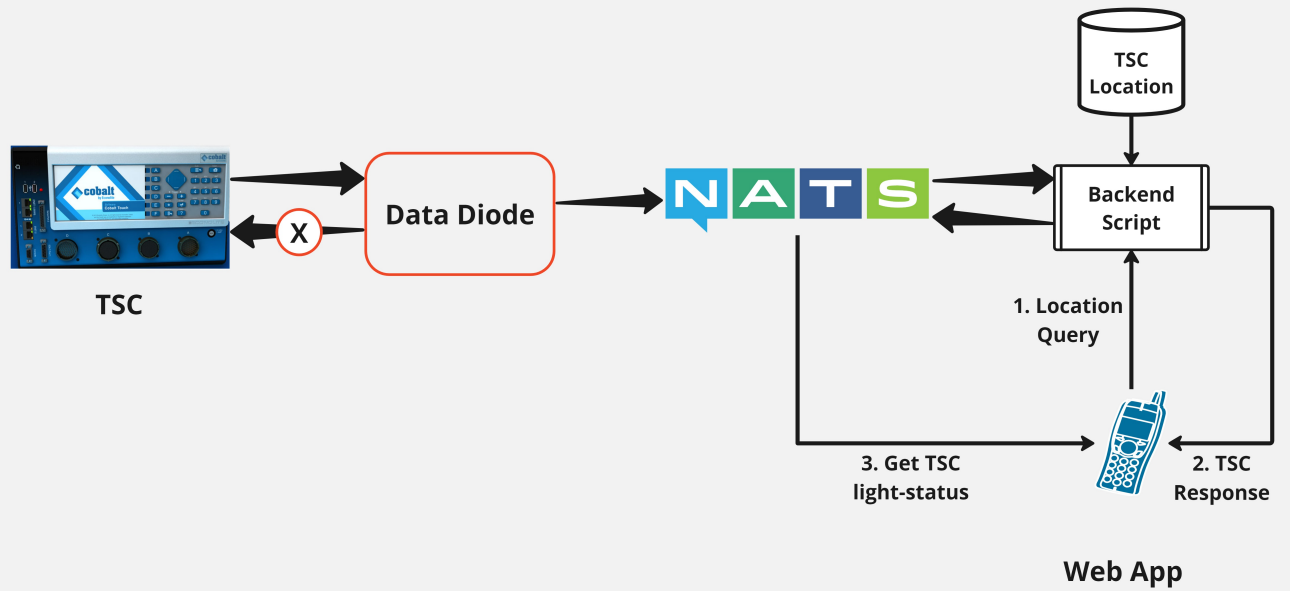

Figure 4: System Workflow

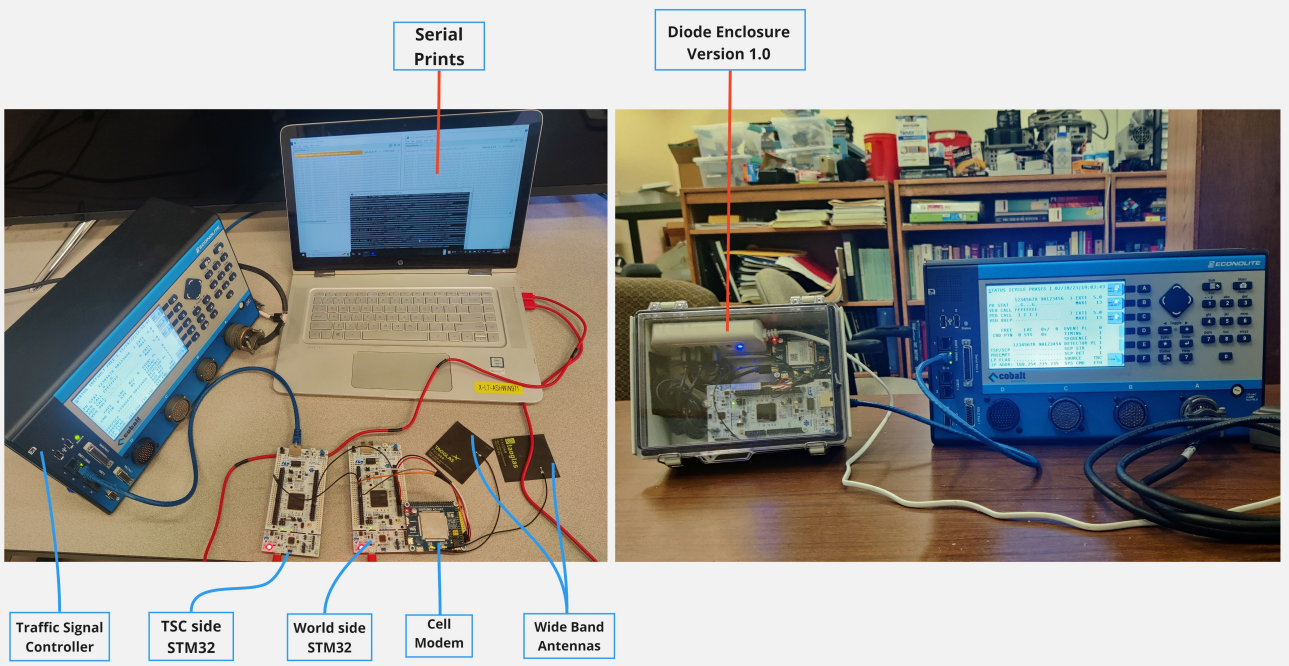

Figure 5: Data Diode Practical Representation

3. Software Code

The comprehensive suite of software and design resources employed in the Data Diode project encompasses a range of essential components, each serving a distinct purpose within the system architecture. This amalgamation of tools and scripts, along with their respective functionalities, is detailed below:

1. **STM32 Arduino IDE Programs:** The project features intricately crafted STM32 Arduino Integrated Development Environment (IDE) programs, catering to both the Traffic Signal Controller (TSC) side and the World side of the Diode. These programs are meticulously designed to facilitate the efficient exchange of SPaT data. Moreover, the Modem Client file, ingeniously integrated within these programs, empowers the establishment of a connection and the seamless publication of data to the NATS server.

2. **NATS Python Script:** A pivotal Python script is incorporated into the system's architecture to proficiently process the transmitted data on the server side. This script adeptly manages the incoming data, assuring its accurate distribution and appropriate utilization.

3. **Svelte Mobile App Code:** The culmination of this project is encapsulated within the Svelte mobile application, meticulously designed to provide an intuitive interface for users. This app efficiently captures and processes GPS location data, enabling users to promptly access real-time intersection information.

4. **Python Program for MAP Conversion:** The system further leverages a Python program adept at processing MAP files and seamlessly converting them into JSON format. This JSON format is not only more accessible but also con-

ducive to efficient storage within the Postgres database, ensuring streamlined data management.

5. **KiCad Files for PCB Layout (Version 2.0):** The system's evolution is evident in the incorporation of KiCad files that meticulously outline the PCB layout for Version 2.0 of the data diode. This advanced iteration builds upon the foundational Version 1.0, culminating in a more refined and optimized design.

6. **CAD Files for Enclosure (Version 2.0):** The CAD files present within the repository are instrumental in the realization of the data diode's physical enclosure. By providing detailed specifications, these files facilitate the construction of a robust and secure housing for the system.

7. **Auxiliary Tools:** Complementary resources, such as "tsc_simulator.py," and pcap files (can be accessed using Wireshark Software) containing SPaT data have been thoughtfully included to simulate TSC behavior in scenarios where direct access to a TSC might be impractical. Additionally, "latency_measurement.py" is a valuable inclusion, facilitating the quantification of system latency for comprehensive performance evaluation.

The entirety of these resources, collectively contributing to the success of the project, is housed within the GitHub repository accessible at https://github.com/oats-center/data-diode. Within this repository, stakeholders can access, examine, and engage with the various tools and components integral to the Data Diode project.

These multifaceted resources collectively underpin the functional prowess of the Data Diode project. The repository's accessibility ensures that stakeholders and enthusiasts can readily explore, understand, and harness the capabilities of this innovative endeavor. For an in-depth exposition on the specifics of each resource, their roles, and their interplay within the project, Chapter 4 of this report serves as an informative guide.

4. Approach

The holistic connection framework has been elaborated upon in Chapter 2. A comprehensive comprehension of the overarching system can be achieved through a meticulous exploration of the diverse components elucidated in Chapter 2. This current chapter further elucidates these integral elements to provide a comprehensive grasp of the entire system's intricate workings.

4.1 The Controller side of the Diode

The TSC side of the Data Diode is effectively realized through the execution of four distinct sub-tasks, which are comprehensively outlined in Chapter 2. The subsequent sub-sections elaborately delve into the intricacies of each of these four sub-tasks, collectively shedding light on their integral roles within the overarching system.

4.1.1 Programming the TSC

Our engagement encompassed two distinct controllers: the ATC eX NEMA Controller and Econolite's Cobalt ATC Traffic Controller. The McCain controller was secured through the invaluable assistance of Dr. Darcy Bullock, who occupies the esteemed role of Lyles Family Professor of Civil Engineering. Regrettably, the McCain controller necessitated a software update to enable it transmit SPaT data through its Ethernet ports. To address this requirement, the McCain support team advocated shipping the controller to them for the update. However, this approach posed challenges due to the absence of a definitive timeframe for resolution from the McCain Support team.

Consequently, we pivoted towards procuring another Econolite controller, this time through the Turner-Fairbank Highway Research Center (TFHRC). This alternative Econolite controller emerged as a viable solution for transmitting SPaT data. In a notable display of collaboration, the TFHRC personnel provided pivotal guidance in programming the Econolite controller. This collective effort not only facilitated the accomplishment of our objectives but also underscores the significance of collaborative endeavors in the research landscape.

To effectively enable the Econolite Traffic Signal Controller (TSC) for transmitting SNMP-based data on its Ethernet port, a meticulous programming process is imperative. This programming endeavor is organized into a step-by-step sequence, further divided into two distinct sub-parts:

4.1.1.1 Programming the TSC to Transmit SPaT Message from ENET1 Port: The first phase involves configuring the TSC to efficiently transmit SPaT (Signal Phase and Timing) messages via its ENET1 port. This segment of programming necessitates the following steps:

1. Press "Main" button. This will display "MAIN MENU" option
2. In the Main Menu, press "1" button on the keypad to select "CONFIGURATION" option
3. CONFIGURATION sub menu will be displayed. Press "2" button on the keypad to select "COMMUNICATIONS" option
4. COMMUNICATIONS sub menu will be displayed. Press "1" button on the keypad to select "ETHERNET" option
5. Now Network parameters can be set using the keypad. Once the parameters are set, press "Enter" button.

4.1.1.2 Configuring SPaT Message Transmission on TSC ENET1 Port: The subsequent stage involves configuring the TSC to transmit SPaT unicast messages on its ENET1 port. This configuration is achieved through the following steps:

1. Connect the TSC's ENET-1 port to a PC having an Ethernet port using an Ethernet cable
2. In the PC command(cmd) window type the following cmd and hit enter.

```
$ snmpset -v 1 -c public 169.254.235.235:501
   1.3.6.1.4.1.1206.3.5.2.9.44.1.1 i 2
```

A response will be generated, indicating the successful execution of the SNMP SET command. This response will confirm the setting:

```
SNMPv2-SMI::enterprises.1206.3.5.2.9.44.1.0 = INTEGER: 2
```

In situations where executing the snmpset command encounters difficulties on a Windows system, an alternative approach is available. The utilization of a tool like the MIB Browser, accessible at https://ireasoning.com/mibbrowser.shtml, can offer a solution. This browser serves as a versatile platform that facilitates interactions with SNMP devices and MIB (Management Information Base) objects.

Upon completing these programming stages, the Econolite TSC will be primed to initiate the transmission of data via its ENET1 port, effectively contributing to the realization of the Data Diode system's data acquisition process.

The procedural steps outlined above are succinctly visualized in Figure 6, offering a quick-reference overview of the configuration process. Additionally, a sample network configuration for the same is presented in Figure 7, providing a tangible representation of how the elements connect within the context of the procedure.

For seamless integration, it's crucial to synchronize the **"IPAddress ip"** parameter in the **TSC_Side_STM32.ino** Arduino IDE code with the specific **"SERVER IP"** address inputted during step 5 of 4.1.1.1. More details on Econolite TSC can be found in the "Econolite_Manuals" folder in github

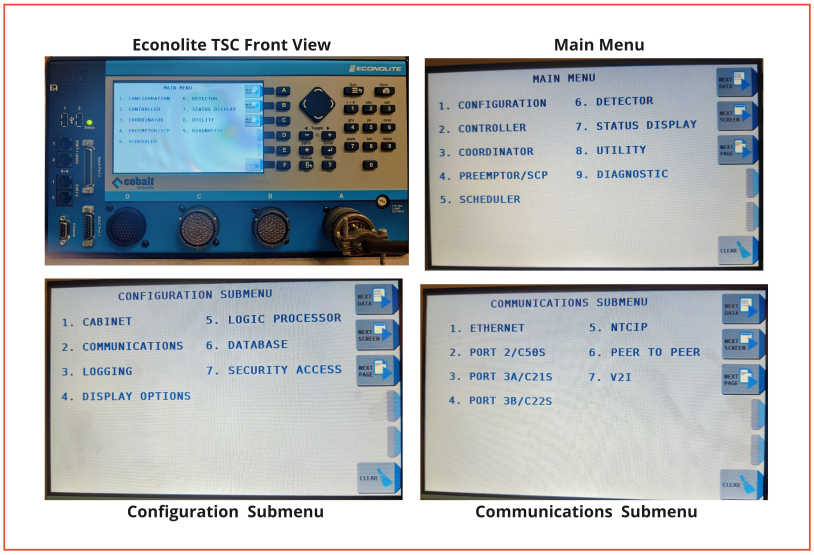

Figure 6: Econolite Display Settings

Subsequent to the initial deployment of the diode at MDOT Signal Shop at Lansing, a pertinent discovery emerged – MDOT had already allocated the ENET1 network for their internal requirements. Consequently, a need arose to reconfigure the Econolite controller to transmit SPaT data via the ENET2 network, instead of ENET1. To effectively reconfigure the transmission route, adhere to the following steps:

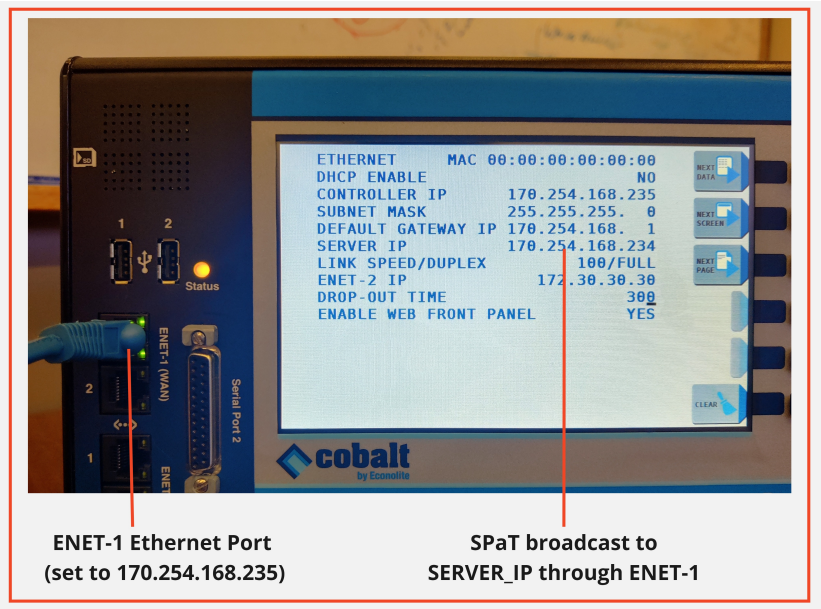

Figure 7: Sample Network Settings for Broadcasting on ENET-1

1. Connect to the controller via ENET-1 and SSH to it
   - ssh *econolite@<ipAddress>*
   - password: ecpi2ecpi

   Here *ipAddress* is the CONTROLLER IP (i.e. ENET-1 IP) in $CONFIGURATION \rightarrow COMMUNICATIONS \rightarrow ETHERNET$ display
2. Run the following commands to edit the network interfaces file:
   - su (enter the root password "*1St0p$h0p*")
   - vi /etc/network/interfaces
3. Edit the file and add your desired interface for eth0:
   - To edit, type: <i>
   - To stop editing, press: <Esc>
   - To save and exit, press <Esc> then type <:wq>
   - To exit without saving, press <Esc> then type <:q!>
4. Example interfaces file settings with eth0(i.e. ENET2) is set to 170.254.168.230 is shown below :

```
auto lo
iface lo inet loopback
# Configure eth0
auto eth0
iface eth0 inet static
address 170.254.168.230
netmask 255.255.255.0
gateway 170.254.168.1
# Configure eth1
auto eth1
iface eth1 inet static
address 169.254.168.229
netmask 255.255.255.0
gateway 169.254.168.1
```

5. using $*ip a*, Check if eth0 is down. If it is indeed down, bring it up using $*ip link set dev eth0 up* cmd. This is shown in Figure 8

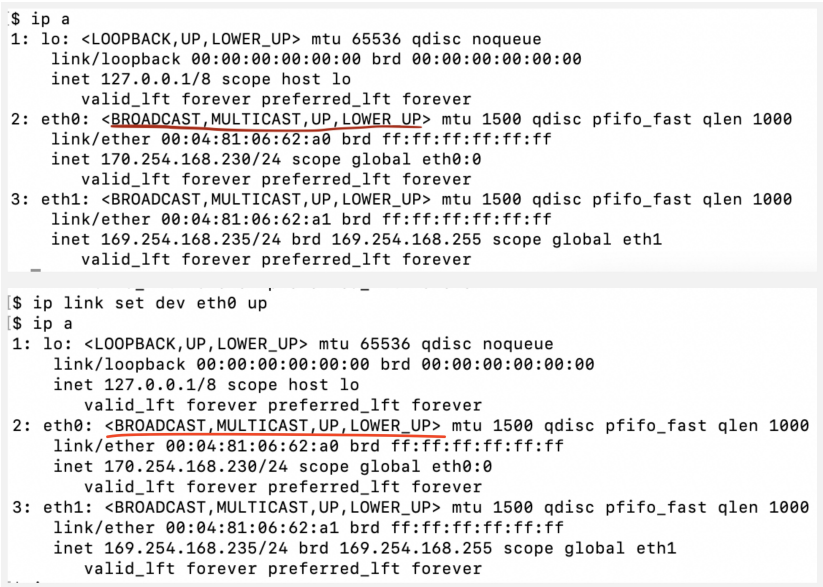

Figure 8: Sample "$ip a" output

6. using $*ip route*, Check if eth0 route is set. If it is not set, set the route using $*ip route add <subnet> dev eth0* cmd. This is shown in Figure 9
7. Set server IP in the communication panel to Diode IP
8. Push the same SNMP command from 4.1.1.2 with the ip address replaced by ENET-2 IP (170.254.168.230 in the sample settings). The data should now flow from ENET2 to destination IP address

Note that only after activation and configuration of eth0 and the corresponding route, one can ssh to econolite using enet2 ip (170.254.168.230 in in the sample settings). This is the reason why CONTROLLER IP(i.e. ENET1 IP) was used to ssh the controller in Step 1.

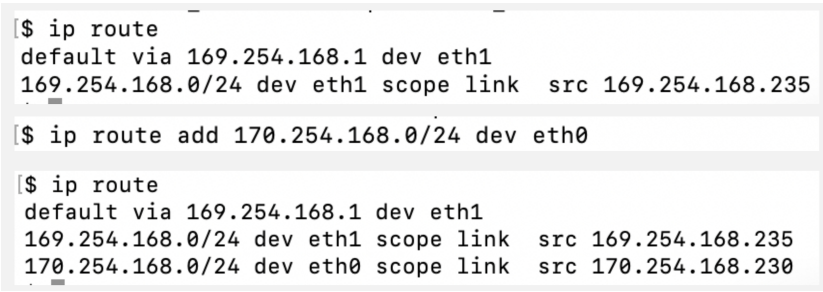

Figure 9: Sample "$ip route" output

However, it's essential to anticipate potential scenarios where the Department of Transportation (DOT) might not grant permission for external entities to modify the SERVER IP. Deploying our diode while setting the IP to the DOT-configured SERVER IP is not a viable solution, as it has the potential to disrupt their network configuration.

One proposition presented by Purdue ECE was to utilize traffic mirroring through iptables. This entails employing the command:

```
iptables -t mangle -A PREROUTING -d <SERVER_IP> --protocol
   udp --destination-port 6053 -j TEE --gateway <DIODE_IP>
```

For example if the SERVER_IP is set as 169.254.168.234, with traffic flowing from ENET-1, and the ENET-2 is on subnet 170.254.168.0/24, then

```
iptables -t mangle -A PREROUTING -d 169.254.168.234 --
   protocol udp --destination-port 6053 -j TEE --gateway
   170.254.168.234
```

will render all the data flowing to 169.254.168.234 through ENET-1, mirrored to 170.254.168.234 on ENET-2. Now we can set 170.254.168.234 as "IPAddress ip" parameter in the TSC Side STM32.ino Arduino IDE code

In summary, the command adds a rule to the PREROUTING chain of the mangle table (The PREROUTING chain in the mangle table allows us to manipulate packets before they undergo the normal routing process), matching incoming UDP packets with a specific destination IP address and port. The rule then duplicates those packets and sends a copy to the specified gateway IP address (the DIODE_IP 170.254.168.234) using the "tee" target However the kernel that the controller is running is a custom built linux which was not compiled with the feature CONFIG_NETFILTER_XT_TARGET_TEE. This feature is what supports traffic mirroring. Perhaps If later iterations of the Project if we could get in touch with Econolite makers they could provide a solution. Note that even if one could successfully configure the mirroring functionality, there remains a potential risk that DOT may prohibit modifications to preconfigured tables of this nature.

An alternative approach to address this issue involves the utilization of a hub that broadcasts data to its output ports, with one of these ports connected to the Diode. Moreover, contemporary switches offer the capability of port mirroring, allowing for data replication to a designated port. This solution can be pursued if an economical

managed switch with these functionalities can be sourced. The solution utilising a hub is shown in 10.

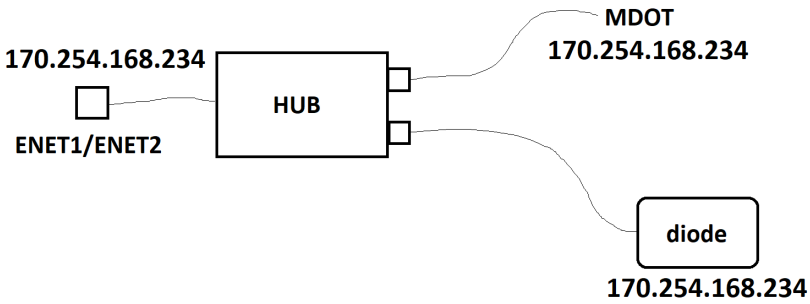

Figure 10: Data Broadcasting via Hub as a Potential Solution

Interestingly, during discussions with MDOT, a potential resolution emerged. They communicated that the ENET2 network wasn't in active use, and there was a possibility of granting permission to employ the SERVER IP for SPaT data transmission. Given this promising development, the detailed exploration of the ipmirroring/hub solution was not extensively pursued.

When engaging in SSH or sending the snmpset command, it's important to bear in mind that the Ethernet IP address of the PC must reside within the same subnet as that of ENET1/ENET2. One can change the Ethernet IP address in the network settings of the PC. This is shown in Figure 11.

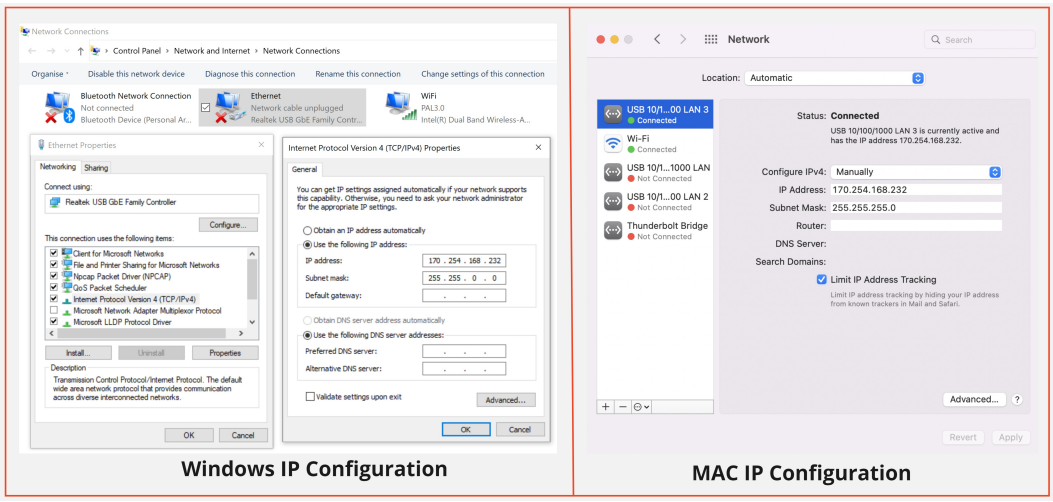

Figure 11: Ethernet IP Settings

Further, one can manually set the PC ethernet to the SERVER_IP and sniff the SPaT data through Wireshark. This is shown in Figure 12.

4.1.2 Programming the STM32 microcontroller to receive SNMP data from the TSC via the Ethernet port

For the implementation of the Data Diode, the NUCLEO-F767ZI STM32 Nucleo-144 Development Board by STMicroelectronics was selected to serve as both the Controller

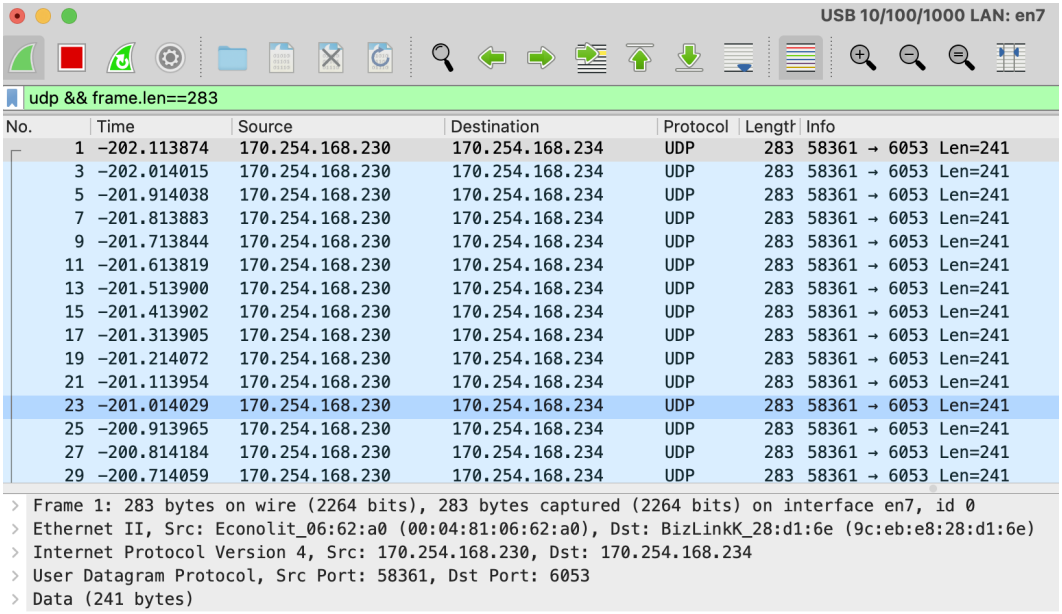

Figure 12: Data Sniffed and Filtered in Wireshark
(Note that the PC Ethernet is set to SERVER_IP)

side and the World side components. This choice was motivated by several factors including its affordability, priced at approximately $23 per board, the presence of an Ethernet port, and the robust support available within the community.

The NUCLEO-F767ZI STM32 Nucleo-144 Development Board was well-suited to fulfill the requirements of the Data Diode project. The inclusion of an Ethernet port within the board facilitated seamless integration into the network architecture, without the need for an external ethernet module. The presence of multiple UART ports on the controller allows to dedicate separate ports for programming the serial connections of both the diode and the modem as well as debug serial prints. Further, it features different colored LED lights integrated within the controller. These LEDs were utilized to provide visual indicators of the system's status, making it easy for observers to monitor the operational state of the system at a glance. Aditionally, the competitive price point ensured cost-effectiveness without compromising on functionality.

For programming purposes, the C++ based Arduino Integrated Development Environment (IDE) was utilized to code both STM32 boards. However, it's worth noting that alternatives such as STM32CubeIDE or Keil, alongside other IDEs compatible with STM32 devices, could equally serve as suitable programming platforms.

The Appendix A.3 of the report contains references to the product website and user manual of the NUCLEO-F767ZI STM32 Nucleo-144 Development Board. This comprehensive documentation serves as a valuable resource for comprehending the board's specifications, capabilities, and programming intricacies.

Following procedure needs to be followed to set up Arduino IDE:

1. In Arduino IDE install "STM32duino_STM32Ethernet" library from $Tools \rightarrow ManageLibraries \rightarrow LibraryManager$

2. Now Click *"File → Preferences → Settings "Additional Boards Manager URLs"*
3. In the *"Additional Boards Manager URLs"* Dialog box select *"Click for a list of unofficial board support URLs"*. This will redirect to a list of 3rd party boards support URLs in github.
4. In this site find the URL for STM32 board by searching for "STM32 core" string
5. Copy this URL and paste it back in *"Additional Boards Manager URLs"* Dialog box in Arduino IDE. One can also paste "STM8 core" URL from the site after the STM32 URL in the same dialog box
6. Next, install "STM32 MCU based boards by STMicroelectronics" from *Tools → Board → BoardsManager*
7. install STM32 Virtual COM Port Driver(STSW-LINK009) from stm site: https://www.st.com/en/development-tools/stsw-link009.html. Once the driver is installed, when a nucleo board is plugged into the system, the Virtual com port for the board will be visible in the device driver list in Windows OS.
8. Now, the COM Port of the STM32 Nucleo 144 board will be visible in *Tools → Port → SerialPort*
9. Select Nucleo-144 under "Boards" in "Select Board → Select Other Board and Port" option

The above steps are shown in Figure 13, Figure 14 and Figure 15.

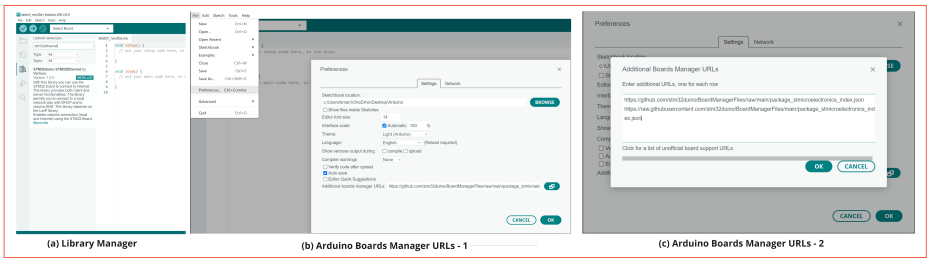

Figure 13: Arduino Library and Boards Manager URLs

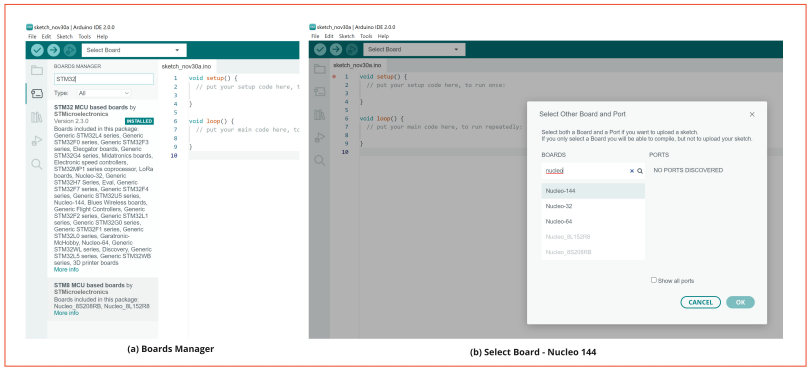

Figure 14: Arduino Boards Manager

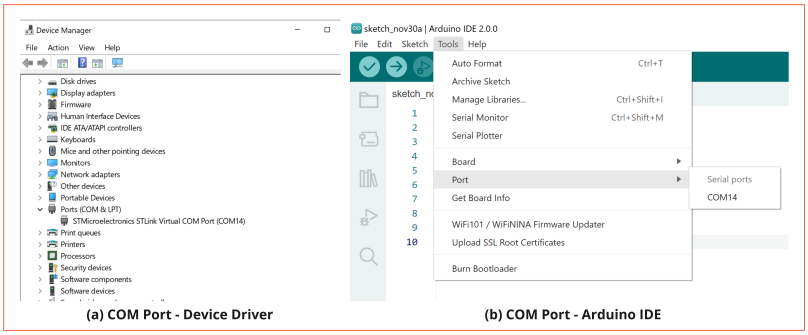

Figure 15: COM Port View

Now *File*− > *Examples*− > 01.*Basics*− > *Blink* can be uploaded to check if the setup is working fine. If Blink program successfully runs on the Nucleo-144 board, "TSC_Side_STM32.ino" file in the git repository of section 3 under the path "DATA_DIODE\Data_Diode_efficient_heap_based\TSC_Side_STM32 can be uploaded after setting the

```
IPAddress ip(170, 254, 168, 234);
```

object in the file to be same as the server address set in the TSC as explained in subsection 4.1.1. In the reference file, the object is set to IP address 170.254.168.234 with 4 arguments representing the 4 octets of the IPv4 address. Further subnet parameter and SPaT broadcast port can to be changed as per the network settings of the TSC. In the file, these values are 255.255.255.0 and 6053 respectively. By default the TSC transmits SPaT to UDP port 6053. This can be checked by sniffing the data through wireshark. Also, its important to have both the Diode IP and ENET IP on the same subnet. Serial prints have been kept bare minimum for maximum transmission efficiency. When the configured TSC is connected to the Ethernet port through an Ethernet cable, an indicator light starts blinking at approximately 100ms intervals. This blinking serves as an indication of the successful transmission of data over the network connection.

4.1.3 Encoding data using a Base64 encoder, including the received SPaT data and a CRC value

In the data processing pipeline of the system, an important step involves ensuring data integrity and converting the binary data into a format suitable for transmission and storage. To achieve this, a 32-bit Cyclic Redundancy Check (CRC) is applied to the incoming data.

The CRC is a type of error-detection code that is calculated based on the content of the data itself. It is used to detect any alterations or errors that may have occurred during the transmission or storage of the data. By appending this CRC value to the original data, the recipient can verify whether the received data matches the originally transmitted data.

CRC32 arduino module available at https://github.com/RobTillaart/CRC was used for calculating the CRC of the incoming(from TSC) SPaT data. Once the CRC

value has been added to the data, the combined information is encoded using a Base64 encoder plugin. Base64 encoding is a binary-to-text encoding scheme that represents binary data as a sequence of printable ASCII characters.

Once the CRC value has been added to the data, the combined information is encoded using a Base64 encoder plugin. Base64 encoding is a text encoding scheme that represents binary data as a sequence of printable ASCII characters. It is widely used for data transmission over networks that may not support binary data directly.

Base64 encoding offers a significant advantage over other encoding schemes with higher overhead, such as Base85 or Hexadecimal encoding. These schemes introduce more characters or bits per encoded unit of data, leading to increased output sizes. Additionally, direct ASCII encoding is not feasible in this context due to the nature of the SPaT data, which can include non-printable digits. These non-printable digits make direct ASCII encoding unsuitable for transmission.

The combination of applying a 32-bit CRC to ensure data integrity and then encoding the data using the efficient Base64 scheme allows for secure and compact transmission of binary data, making it suitable for the communication requirements of the system.

4.1.4 Transmitting the encoded data over the Tx port of a simplex UART connection

Once the programmed STM32 from subsection 4.1.2 is connected to the configured TSC of subsection 4.1.1, one should be able to see the serial prints in the Arduino IDE Serial Monitor. A sample view of this serial prints is shown in Figure 16. In the design of the Data Diode, a crucial aspect is the establishment of a one-way data transmission path between the Controller side and the World side of the Diode. This unidirectional communication ensures that data flows from the Controller side to the World side without allowing any reverse data transmission.

To achieve this, a data cable is used to connect specific pins on the STM32 microcontrollers on both sides of the Diode. Specifically, the Tx (Transmit) port pin PE8 of the STM32 on the Controller side is connected to the Rx (Receive) port pin PE7 of the STM32 on the World side. This connection facilitates the transmission of data from the Controller side STM32 to the World side STM32.Importantly, no direct connection is established between the Rx pin of the STM32 on the Controller side and the Tx pin of the STM32 on the World side. This intentional lack of connection between these pins ensures that data cannot flow in the opposite direction, thus fulfilling the requirement of a one-way communication path.

The transmit(Tx) pin from the first microcontroller is a UART (Universal Asynchronous Receiver-Transmitter) transmit pin that is programmed to transmit data at a specific bit rate. A receive(Rx) pin on the second microcontroller is programmed to listen to this data at the same bit rate. In general UART communication protocol, there are two pairs of such pins, with both devices able to send to and receive data from the other device. However, in the data-diode system, the second link is removed so that only the first microcontroller can transmit data to the second and not vice-versa. It is important to mention that the transmit or receive functionality of a specific pin on

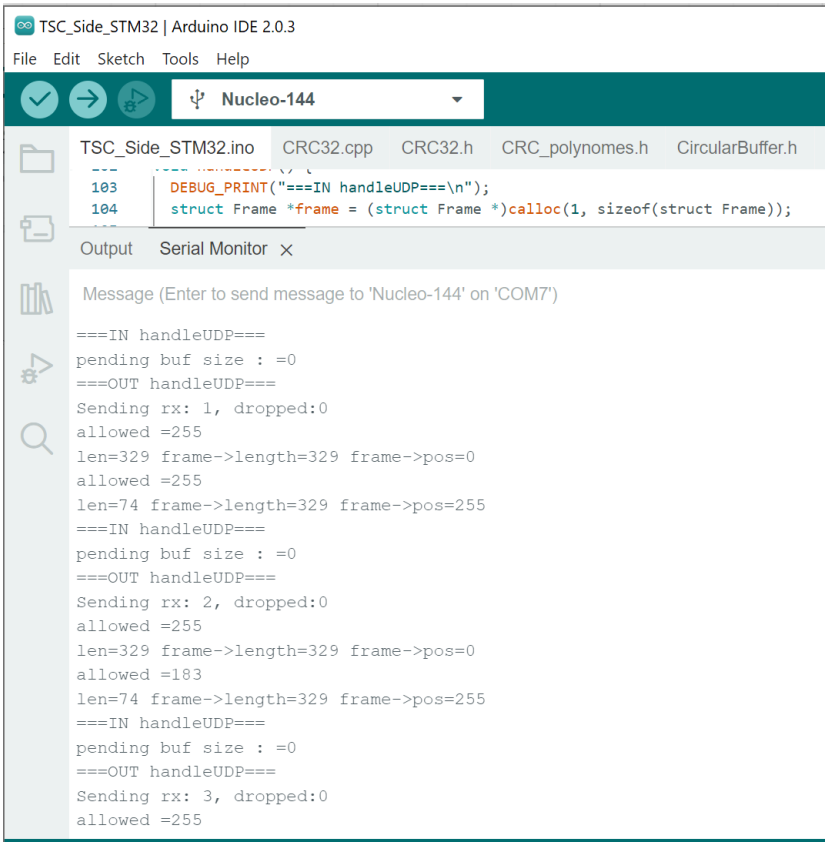

Figure 16: Minimal Serial Print Logs from TSC_Side_STM32.ino File

the microcontroller is fixed and cannot be reversed. i.e., a Tx pin cannot be made to receive data like a Rx pin; and a Rx pin cannot be made to transmit data like a Tx pin even we modify the software. So, there is no data flow from the second microcontroller to the first one as the second Tx-Rx physical link is itself removed in the design. This is shown in Figure 17.

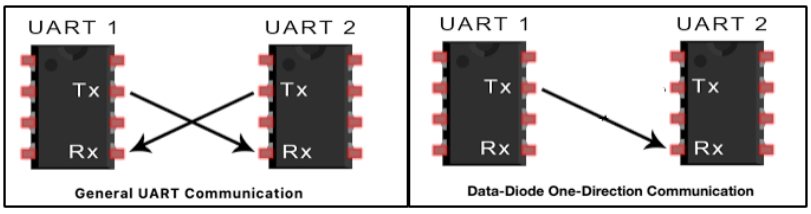

Figure 17: Data Diode One-Direction Communication

In essence, the setup of the data cable and the specific connections made between the Tx and Rx pins on each side of the Diode create a simplex transmission arrangement. This arrangement ensures that data can be sent from the Controller side to the World side, while preventing any potential data transmission in the reverse direction, which is a fundamental characteristic of the Data Diode's operation that isolates the TSC from potential spurious attacks, as shown in Firgure 18

A Baud rate of 115200 was used for the demonstration. The programmer must take care of setting the same Baud rate between the two STM32s. The Tx Ports and

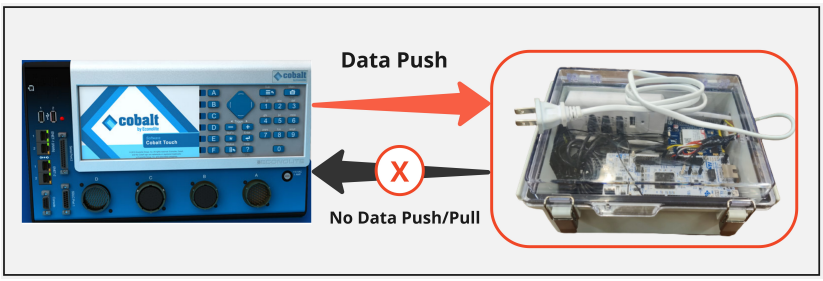

Figure 18: TSC-Diode Data Push Mechanism

their pin mappings for the STM32 Nucleo board can be viewed in the weblink "Pinout" referenced in Appendix A.3.

The graphical representation of the TSC side STM32 code flow is depicted in the Figure 19

If data in transition through the TSC side STM32 needs to be captured on a file or viewed in serial monitor, then either the prints can be introduced in the Arduino file within the UDP interrupt routine or a separate TSC_Side_STM32.ino" file in 3 under the path "Data_Diode_SW_Code\STM32_Program\TSC_Side_STM32" can be uploaded. This version of the code is intended for viewing SPaT data on a serial console, without the addition of CRC and without base64 encoding. The serial data can be further logged in a text file by running "com_stm.py", by setting the appropriate COM Port, baudrate and file name. The Serial prints of this file is shown in Figure 20, while the cmd window output of "com_stm.py" is shown in 21. (Note that for the "com_stm.py" output the UNIQUE_ID prints in the alternate TSC_Side_STM32.ino file has been commented out. If UNIQUE ID is needed, the prints can be retained as well. Alternatively, the UNIQUE_ID string can be replaced by an empty string in "com_stm.py" itself. This alternate TSC_Side_STM32.ino was during the early stages of the project where the UNIQUE_ID captured from TSC side STM32 registers was used for diode identification. In later stages of the project it captured from World side STM32 to increase transmission efficiency). Serial logs of SPaT data and the base64 encoded data from the actual TSC_Side_STM32.ino file is shown in Figure 22. It is of paramount importance to comment out or remove these data print statements, as they have the potential to consume a significant amount of CPU cycles.

4.2 The World side of the Diode

The World side of the Data Diode is realised by executing 3 sub tasks as mentioned in Chapter 2. Each of these 3 sub tasks is explained in the following sub sections.

4.2.1 Programming an STM32 microcontroller to receive encoded SPaT data from the controller side of the STM32 through the Rx port of a simplex UART connection.

The STM32 Nucleo board situated on the "World side" of the diode is also programmed utilizing the Arduino IDE. The programming involves configuring the board to receive data through the Rx port of UART7, which corresponds to pin PE7 on the STM32

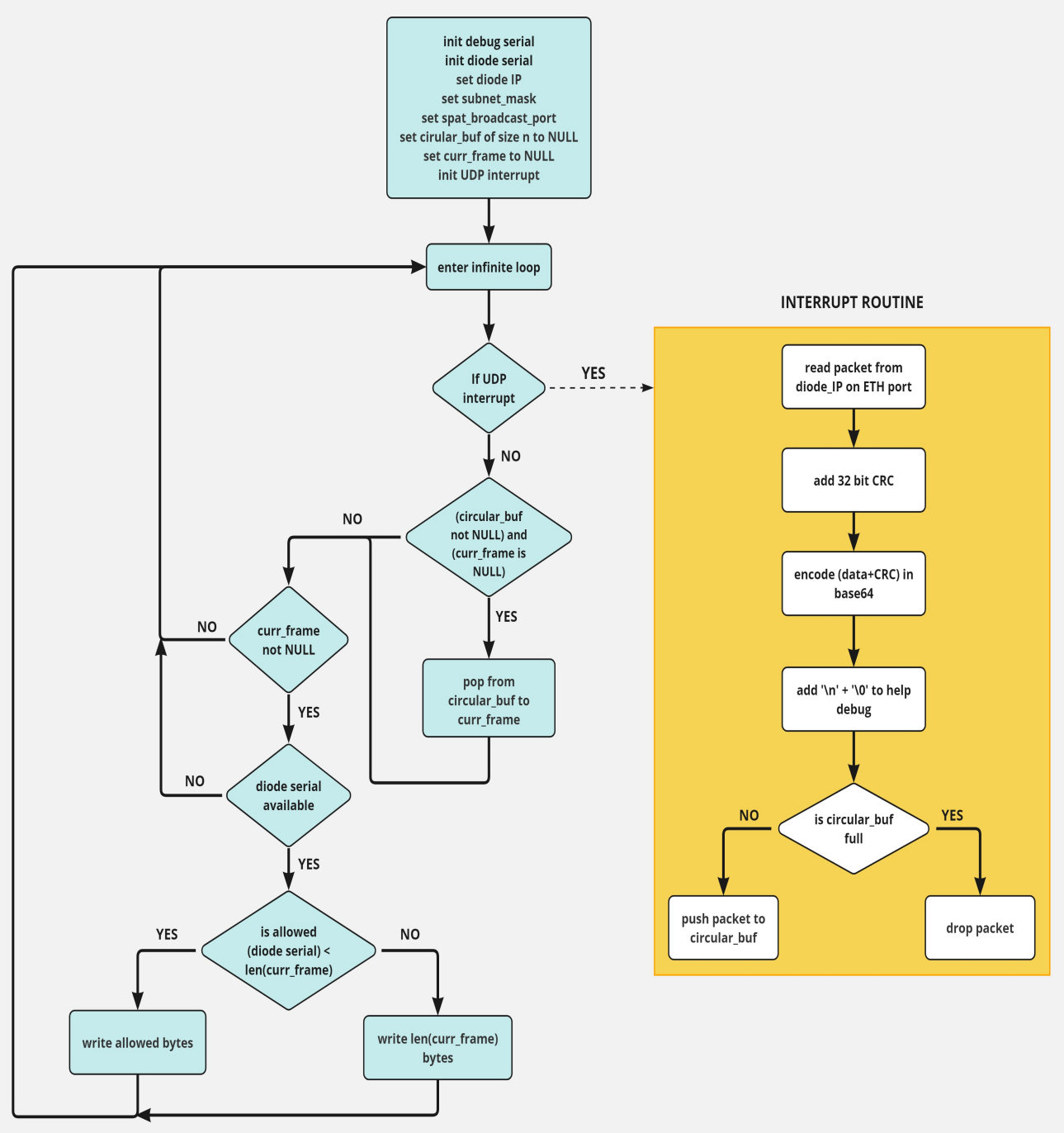

Figure 19: TSC Side STM32 Codeflow

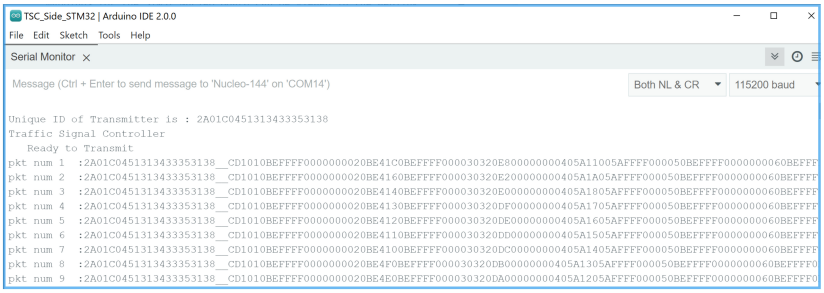

Figure 20: Serial Print Logs from alternate TSC_Side_STM32.ino File

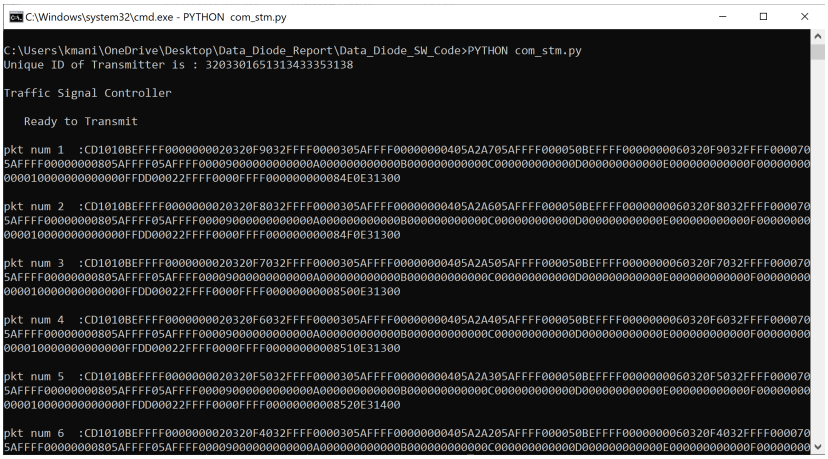

Figure 21: CMD Window Output of "com_stm.py"

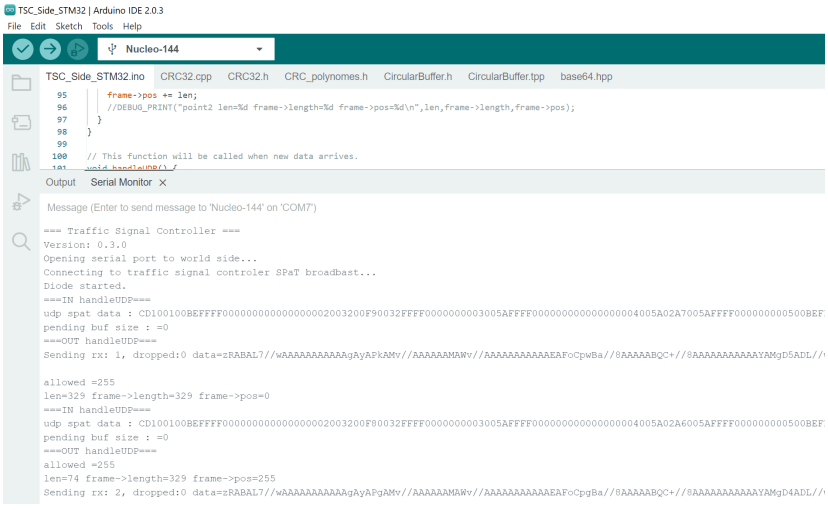

Figure 22: SPaT and base64 Serial Print Logs from TSC_Side_STM32.ino File

microcontroller. During the programming process, minimal modem client prints and data traffic status are logged on the serial monitor. This logging mechanism serves the purpose of providing monitoring information. It's worth noting that similar to the STM32 on the TSC side, these traffic loggings can be disabled on both STM32 microcontrollers by undefining "DEBUG" parameter in the file. This flexibility to disable logging can help conserve CPU processing cycles and optimize the performance of the system. A sample view of the serial data traffic and prints can be observed in Figure 23.

During the operational phase, the serial print of the "***data***" parameter in the line "**New message. Length =**" which represents the incoming data, is intentionally omitted from being printed. This exclusion is implemented to conserve CPU cycles and optimize the processing efficiency of the system. While the serial print statement is used for debugging purposes during development, it is typically removed or commented out in the final operational version of the code. This ensures that the system dedicates its computational resources primarily to its intended functionality, rather than printing debugging information to the serial monitor. To ensure proper communication, it's essential for the programmer to set the same Baud rate for communication between the two STM32 microcontrollers. This synchronization of Baud rates ensures that data can be accurately transmitted and received between the components of the diode, facilitating smooth operation and reliable data exchange.

4.2.2 Interfacing the STM32 on the World side with a 4G LTE-compatible cell modem

The data received by the second STM32 needs to be transmitted to a remote NATS server to facilitate further data processing. To achieve this, a SIM7600X 4G HAT module from Waveshare Electronics was chosen. This selection was based on the module's support for multiple Radio Access Technologies (RATs), the availability of both UDP and TCP-based data transmission capabilities, simple UART-based interfacing, a well-compiled AT command manual, and other advantageous features. Additional information about this modem can be found in the product's wiki page, which is referenced in Appendix A.4.

The communication with the cell modem is established using a range of AT commands. These commands are used to perform tasks such as establishing connections, obtaining and providing network information, and more. A comprehensive set of AT commands can be found in the modem's manual, which is also referenced in Appendix A.4. Specifically, the AT commands outlined in Chapter 11 of the manual were utilized to create a TCP session with the remote NATS server hosted at Purdue University. This TCP session allowed for the transmission of the received data over the diode to the server.

In Figure 26, you can see that UART6, located on the World side of the STM32, is connected to the PG9 and PG14 pins (or D0 and D1 pins) of the SIM7600X 4G HAT modem. These connections establish communication between the STM32 and the modem, with UART6's TxD and RxD pins corresponding to the corresponding pins on the modem.

Furthermore, the D7 pin (PF13 pin) of the Nucleo board serves to control the

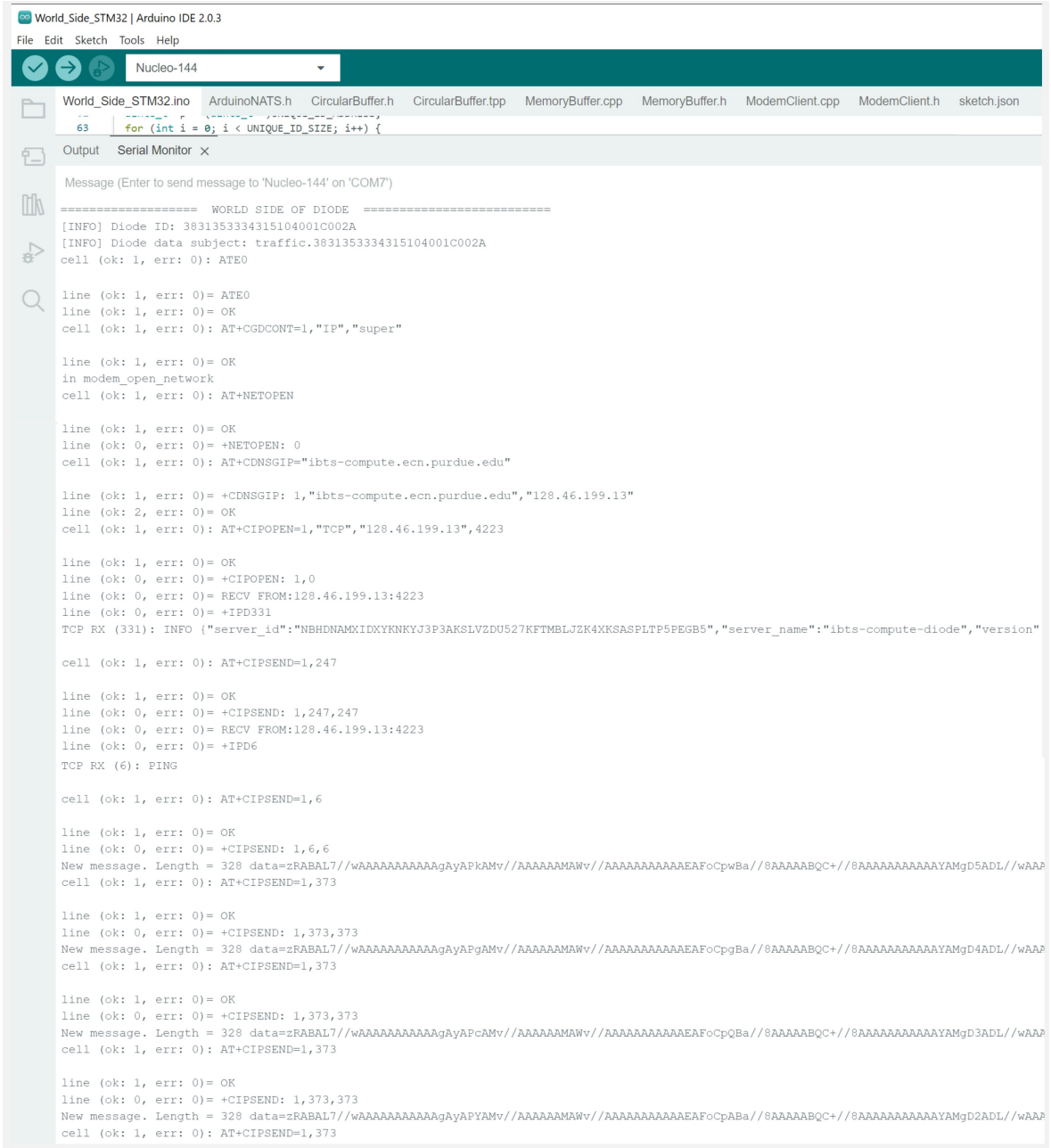

Figure 23: Serial Print Logs of World_Side_STM32.ino File

modem through the PWR pin of the modem. Both the microcontrollers and the modem are powered by the same input power source, typically at 5V. To ensure proper power supply, it's essential to disconnect the jumper between the 3.3V and PWR pins of the modem, especially when 5V is provided as input.

Once a 4G SIM card is inserted and the modem is connected to the network, it will be prepared to transmit data. The NET indicator pin on the modem will blink approximately every 200ms, as outlined in the modem's wiki page. It's worth noting that it might take a few minutes for the modem to successfully register on the network.For effective communication, make sure the UART selection jumper remains in position B, which indicates controlling the SIM7600 through the Raspberry Pi interface. This configuration ensures the proper interaction between the STM32 and the modem.

Two Wide Band Antennas from Taoglas which operate in 4G frequency range were used to connect to the AUX and MAIN Antenna Ports of the cell modem. Although practically the modem can transmit the data even without the antennas, it is advisable not to operate the modem without the antennas as the resulting higher power might impair the RF electronics of the modem. The antenna surface have to be placed at 90 degree angle with respect to the other. The datasheet of the antenna is referenced in Appendix A.4

While it might be possible to transmit data using the modem without the antennas, it's strongly recommended to use the antennas at all times. Operating the modem without antennas could result in higher power levels that may potentially affect the RF electronics within the modem itself.

Placement of the antennas is also essential. The surface of one antenna should be positioned at a 90-degree angle with respect to the other antenna. This arrangement helps optimize signal reception and transmission. For more detailed specifications and information about the antennas, you can refer to the datasheet provided in Appendix A.4.

For establishing communication between the modem client and the NATS server, an Arduino NATS library, **"ArduinoNATS.h"** was utilized, available at `https://github.com/isobit/arduino-nats`. However, since the library was initially designed to be compatible with Ethernet and WiFi-capable devices, a new modem client was developed to interface with the Arduino NATS library. This new client was specifically designed to work with the SIM7600 series modems from SIMCom.

The modem client was crafted using AT commands to facilitate communication with the NATS server. It adopts a state machine model, which means it transitions through various states as it receives information and transmits commands and data to the NATS server. The State machine diagram outlining the different states of this Modem Client can be seen in Figure 24.

This modem client is responsible for several tasks, most imortant of which are :

1. Defining the Packet Data Protocol (PDP) context.
2. Initiating the TCP service.
3. Performing Domain Name System (DNS) lookup.

4. Opening a TCP socket.
5. Transmitting and receiving messages to and from the NATS server.

To ensure the ongoing connection's vitality, the **"ArduinoNATS.h"** library ensures that PING messages sent from the server are acknowledged with PONG responses. This interaction helps to maintain an active connection between the modem client and the NATS server.

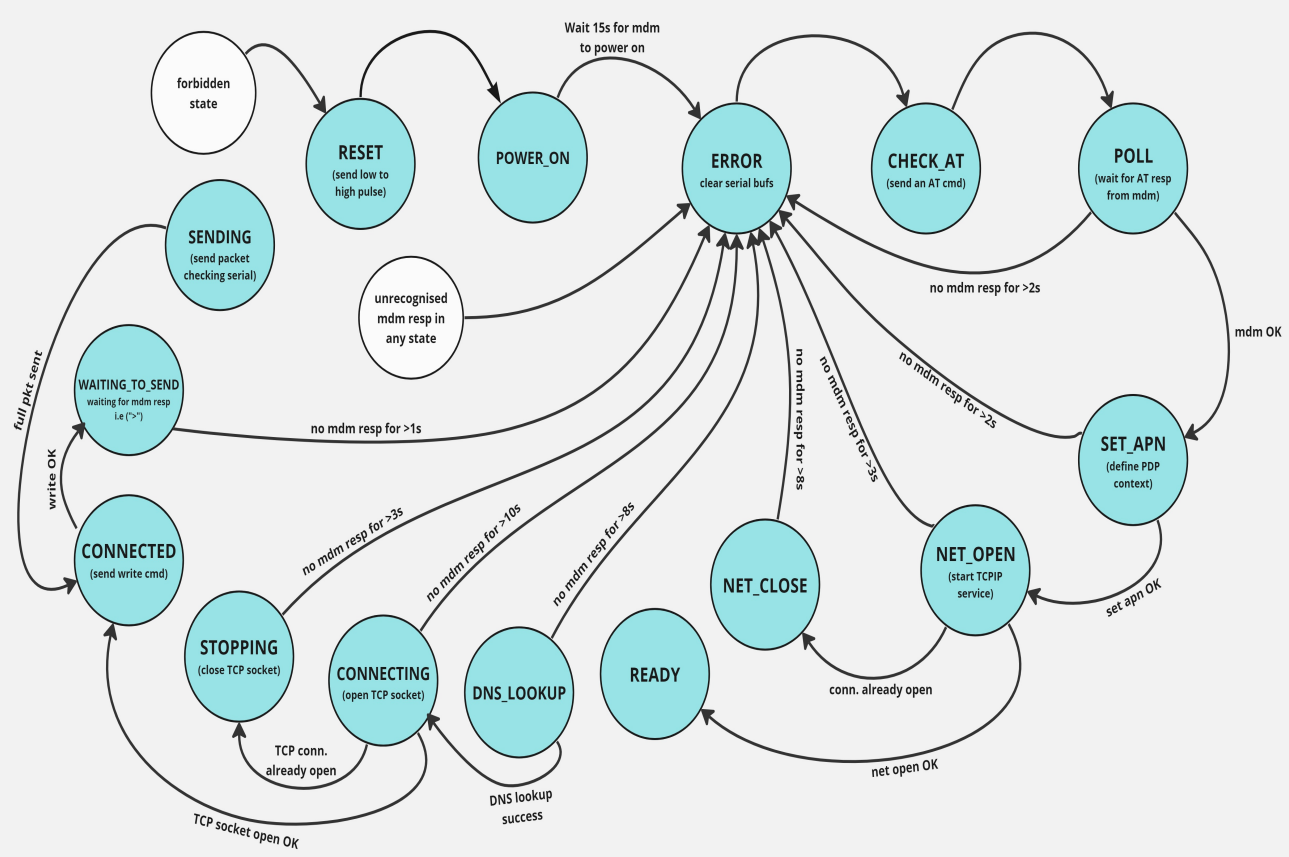

Figure 24: Modem Client State Machine

The dns address and TCP port of the server is input to the nats object created in the file "World_Side_STM32.ino" in the git repository of section 3 under the path "Data_Diode_SW_Code\STM32_Program\World_Side_STM32". The reference code provided for example has NATS address set to ibts-compute.ecn.purdue.edu and the Server being listening on TCP port 4223 i.e.

```
NATS nats(&client, "ibts-compute.ecn.purdue.edu", 4223, "<
    username>","<password>");
```

The <*username*>and <*password*>has to be replaced with the appropriate access info of the NATS server. Once these values are set, the code "World_Side_STM32.ino" can be uploaded to the STM32 Nucleo board.

4.2.3 Publishing the encoded SPaT data to a remote NATS server using the Modem Client, with the data destined to the NATS subject specific to "UNIQUE ID"

The process of data transmission from the TSC side STM32 involves a series of steps that ensure the integrity and identification of the transmitted information. A 12-byte unique identifier (UNIQUE_ID) provided by the STM32 designers is used to uniquely identify a diode (and threfore an intersection). The incoming base64 encoded SPaT data is sent via the cell modem a NATS subject, whose name is constructed based on this UNIQUE ID. In the deployed code, the subject is "**traffic.<UNIQUE_ID>**".

Initially, during the initial stages of the project, the UNIQUE_ID was added to the data directly at the TSC side of the diode. However, as the project progressed, it was found that reading the UNIQUE_ID at the World side STM32 was more efficient. This decision was made to optimize system performance, as the addition of extra bytes to the data at the TSC side STM32 would introduce additional overhead and potentially slow down the system.

Each individual Diode system is transmitting its encoded data to a subject name having its a unique identifier, and this identifier is recognized by the NATS server script. The NATS server script uses this UNIQUE_ID, that can be reasoned from the subject name, to discern the specific physical intersection from which the data originates. This identification process is crucial for correctly processing and analyzing the SPaT data received from different Diode systems, ensuring accurate results for each intersection. More information about the register from which this 12 byte UNIQUE ID is read from can be found in Chapter 45 (Device electronic signature) of the STM32 reference manual cited in Appendix A.3.

Moreover, it is worth noting that the modem may experience occasional disconnections from the NATS server due to a variety of factors, including but not limited to low or absent signal strength from the network operator, encountered error messages, and even uncertain weather conditions. In such instances, the modem will transition out of its CONNECTED state, which is indicated by the illumination of LED2 in red. Subsequently, the modem initiates an automatic reconnection attempt in order to regain connectivity with the NATS server.

Upon successful reconnection, as signified by the LED3 turning blue, a significant feature comes into play. The reason for the most recent disconnection is captured and then published to a subject listening to error messages, which in the code is labeled as "**connect.<UNIQUE_ID>**". This mechanism serves as a valuable tool for monitoring and assessing the modem's interactions with the NATS server. It allows for the identification of specific reasons behind disconnections, such as signal-related issues or external factors like unfavorable weather conditions. Additionally, this functionality aids in gauging the frequency of disconnections, contributing to a comprehensive understanding of the overall reliability and stability of the communication link between the modem and the NATS server.

The graphical representation of the world side STM32 code flow is depicted in Figure 25 and the interconnection of the components described thus far is shown in Figure 26

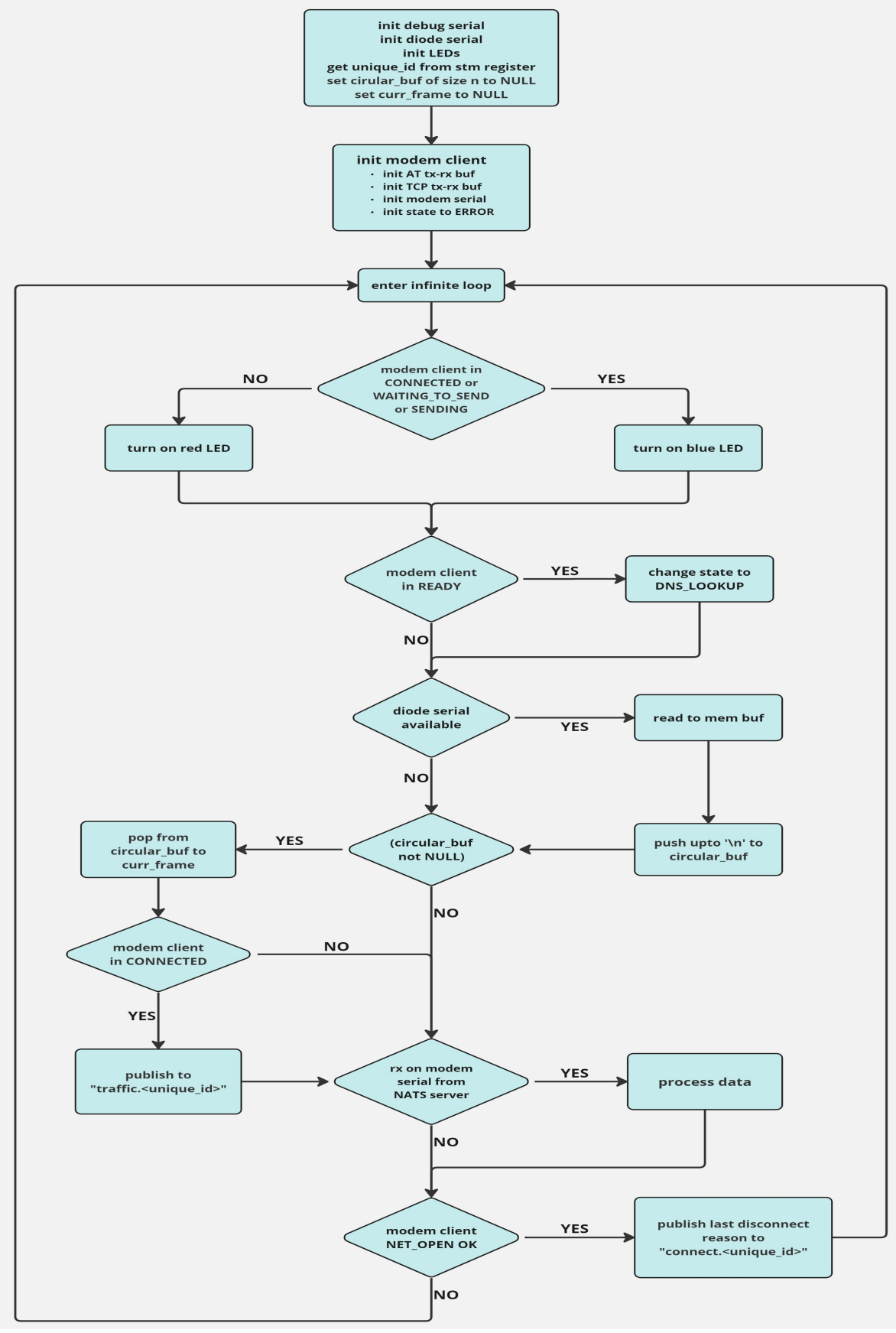

Figure 25: World Side STM32 Codeflow

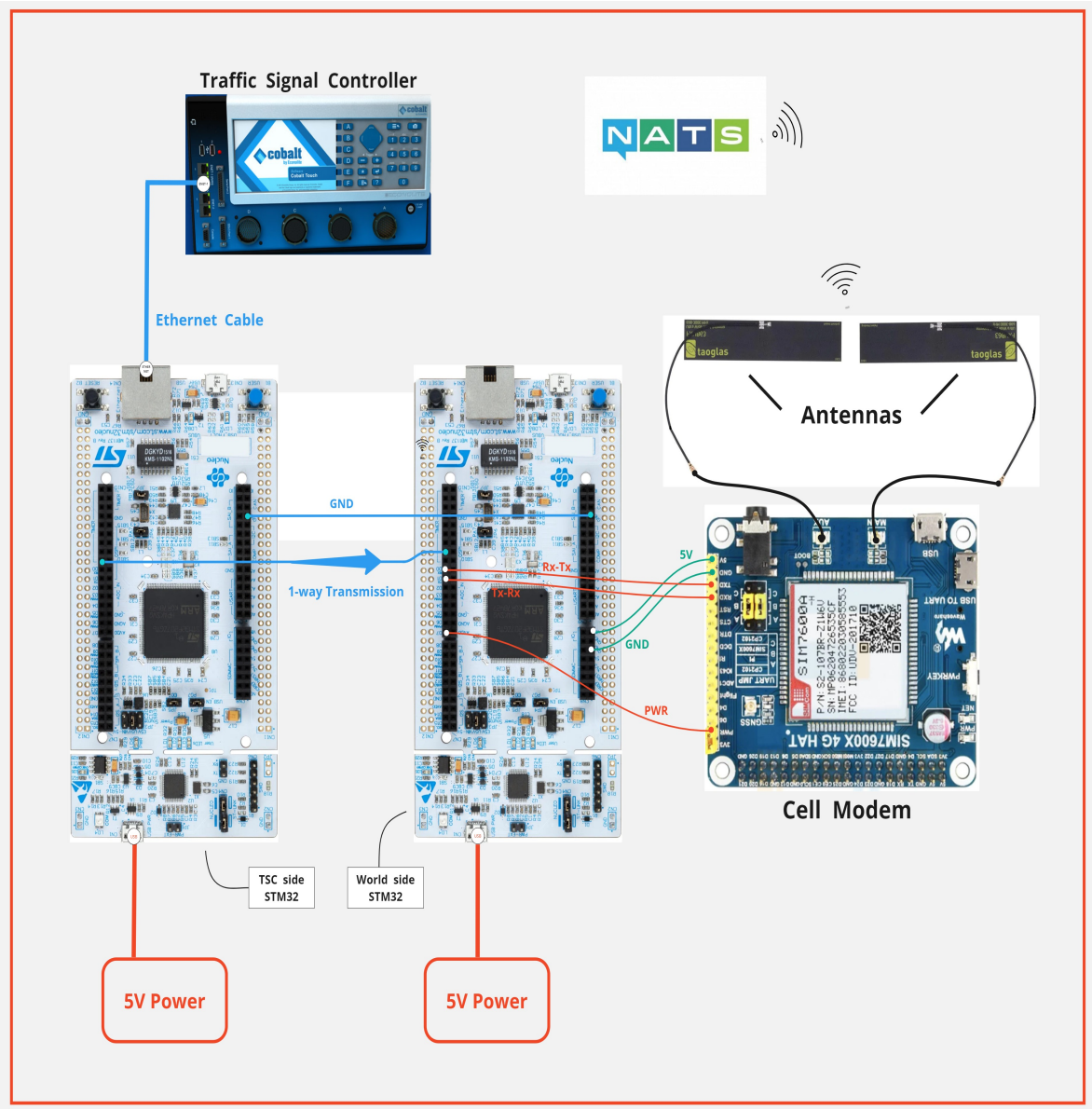

Figure 26: Data Diode Components Interconnection

Similar to the STM32 on the TSC side, if one needs to capture data in transit through the STM32 on the World side and wish to save it in a file or view it on the serial monitor, they have the option of using a separate "World_Side_STM32.ino" file located in the 3 section under the path *"Data_Diode_UDP_Based\STM32_Program\ World_Side_STM32"*.

This version of the code is designed for viewing raw SPaT data on a serial console. If one wish to log the serial data into a text file, they can use the same "com_stm.py" script as mentioned before. The Serial prints generated by this code version are depicted in Figure 27.

It's important to note that unlike the "TSC_Side_STM32.ino" code, if print statements are introduced within the actual "World_Side_STM32.ino" code, the printed output will display the base64 encoded data. This might not be suitable for easily interpreting the transmitted SPaT data.

Figure 27: STM32 Serial Traffic Prints on World side of the Diode from alternate "World_Side_STM32.ino"

Note that the COM Ports of the two STM32s will be different and if code upload is done using the same PC (which will usually be the case), care must be taken to chose the right COM port before hitting the upload button. Also note that the serial data from both STM32s can be monitored on the same PC by opening two instances of Arduino IDEs and listening on different ports. Serial data can also be viewed using other serial console softwares like Putty or Tera-Term. This scenario is depicted in Figure 28. In this setup, the USB cables connecting to the two STM32s serves the dual purpose, mentioned at the start of this section, of supplying input power to the STM32s and providing for a serial interface to monitor the traffic flow.

4.2.4 LED Indications for Data Diode Operation and Cellular Modem Status

During the normal operation of the Data Diode, when it is powered using a standard 5V power source, the LED 1 on the Diode should blink in a red color. As the cellular modem establishes its connection with the network, specifically while the TCP socket is not yet established with the NATS server, the LED 3 will be illuminated in red, and the blue LED 2 will be off.

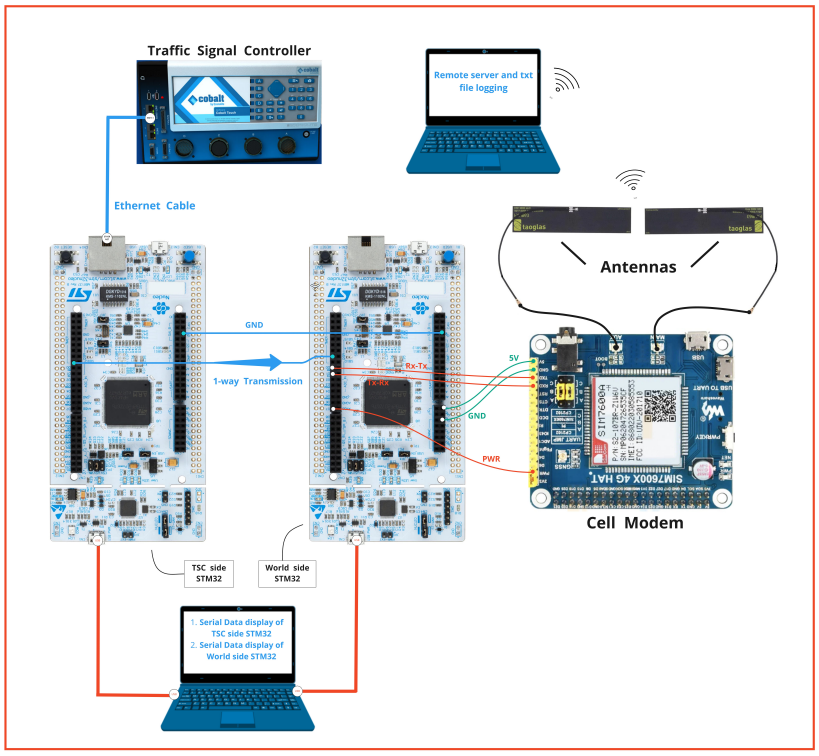

Figure 28: Serial Data Logging

Once the TCP socket is successfully opened and the modem enters the CONNECTED, WAITING_TO_SEND, or SENDING states, the LED 2 will turn blue, and the red LED 3 will be turned off. This LED color indication provides visual feedback about the modem's connection status and activity during the process of transmitting data through the Data Diode system. The Nucleo Board with the labelled LEDs are shown in the Figure 29.

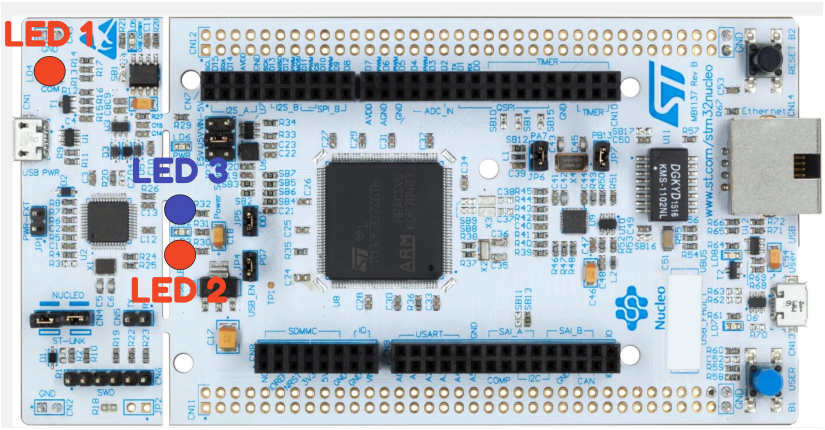

Figure 29: World Side STM32 LED Indicators

4.2.5 Optimizing Serial Buffer Size for Efficient Serial Communication

To ensure optimal performance, it is crucial to configure the **SERIAL_TX_BUFFER_SIZE** and **SERIAL_RX_BUFFER_SIZE** parameters correctly. This configuration can be

achieved by modifying the HardwareSerial.h file

```
#if !defined(SERIAL_TX_BUFFER_SIZE)
#define SERIAL_TX_BUFFER_SIZE 256
#endif

#if defined(SERIAL_RX_BUFFER_SIZE)
#define SERIAL_RX_BUFFER_SIZE 256
#endif
```

The location of this file may vary based on the system, but for the system used for development, it resided at `C:\Users\<username>\AppData\Local\Arduino15\packages\STMicroelectronics\hardware\stm32\2.4.0\cores\arduino`. By applying these settings, the length of the serial transmit (tx) and receive (rx) buffers is set to 256 bytes. Consequently, the STM32 microcontroller can handle a maximum of 256 bytes for both reading and writing when data is available

It is advisable to choose a buffer size that is a power of 2, as this choice significantly optimizes modulo operations for ring buffers. However, it's important to note a potential concern when increasing buffer sizes to values greater than 256. Although the buffer index variables are automatically resized to accommodate the larger buffer size, the additional atomicity guards necessary for this resizing are not implemented. This can lead to the occasional occurrence of a race condition, causing unpredictable behavior in the Serial communication. Therefore, when expanding buffer sizes beyond 256, it's essential to be cautious and monitor the system's behavior closely.

4.3 TSC Simulator

When a TSC is available at user's disposal, the **"tsc_simulator.py"** in the git repository of section 3 in the folder "Data_Diode_SW_Code" can be used to replay data onto the Ethernet port of the Controller side of the STM32 using pre-captured wireshark pcap files. Few PCAP files are already made available in the "pcap_files" directory in the same path. User can input these files by modifying "tsc_simulator.py" accordingly

Few points to be noted while running the "tsc_simulator.py" are:

1. The "localPort" value on "TSC_Side_STM32.ino" file must match with that to which the "tsc_simulator.py" binds
2. The IP address to bind to must be same as the "IPAddress ip" parameter in the "TSC_Side_STM32.ino" file. This IP address inturn must be within the subnet that the laptop Ethernet port exposes. Alternatively one could use x.y.255.255 in "tsc_simulator.py" and broadcast the pkt
3. "allTraffic.pcap" file has many protocol packets(TCP, UDP, HTTP, etc). Only UDP to port 6053 will be received by the STM32 on the controller side of the Diode. This script will be modified to measure the latency of the system, details of which will be provided in Chapter 4.3.

Figure 30 shows the system interconnection while replaying the pcap data onto the

Diode using the "tsc_simulator.py" Figure 31 shows the method to run "tsc_simulator.py" and its sample output.

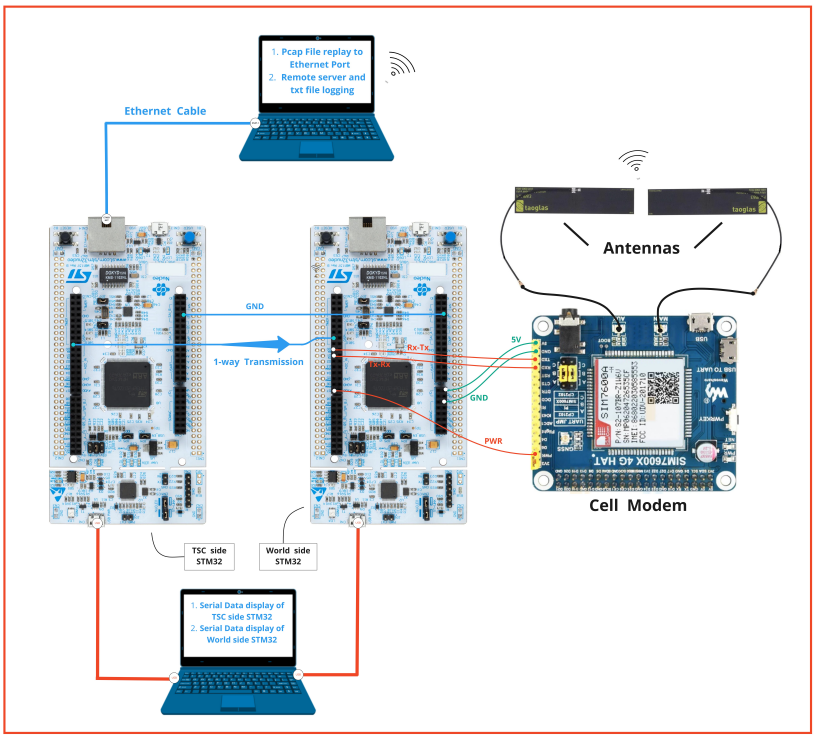

Figure 30: Simulating a TSC by replaying PCAP file to the PC Ethernet Port

4.4 Server/Edge Side of the System

The Server/Edge Side of the system underwent a transition in communication protocols. Initially, a simple UDP server was established to receive and process data from the diode. However, this approach was later upgraded to utilize the NATS open-source high-performance messaging system due to its numerous advantages. NATS offered a more sophisticated and efficient data handling mechanism.

NATS employs subject-based addressing, enabling data to be organized and accessed through subjects. Subjects are sequences of tokens separated by dots. This subject structure allows easy subscription and publication of data between different components of the system. When a server is configured to listen on a specific subject, requests can be made to that subject, and the relevant data is then exchanged.

The decision to adopt NATS was guided by several benefits it provides. Firstly, NATS offers a secure, ultra-low latency, and high-performance data transfer mechanism. Its ease of implementation, comprehensive documentation, and support for NATS client in multiple programming languages also contributed to its selection. NATS enables subjects to serve as points of interaction between the diode and the server, facilitating a streamlined communication flow.

The NATS core introduces subject-topic based request-reply messaging, which is particularly advantageous in the context of multiple diode systems. This approach ensures that the diodes and the NATS server do not need to have direct knowledge

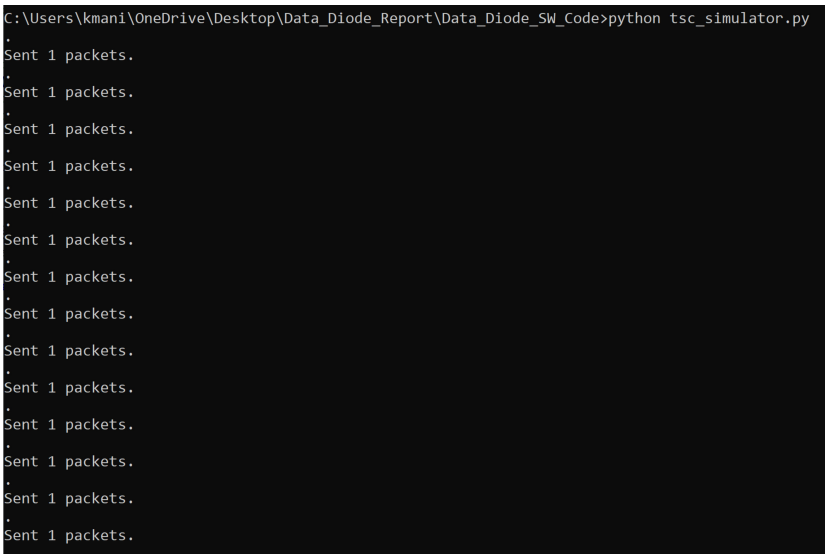

Figure 31: Sample output of **tsc_simulator.py**

of each other's presence. Instead, they communicate through subjects, eliminating the need for establishing specific point-to-point connections. This design stands in contrast to conventional Remote Procedure Call (RPC) technologies, which require a dedicated one-to-one connection between the requester and the replier.

As a testing resource, the demo.nats.io server can be used to simulate and assess the functionality of the NATS-based communication system. Additionally, the NATS command-line interface (CLI) tool proves useful for simulating data diode message packets without requiring the actual diode for testing purposes.

For further information about NATS and its capabilities, interested individuals can visit the official website at https://nats.io/.

4.4.1 Establishing a NATS server configured to continuously listen for data from all TSC-Diode systems

To facilitate the communication between the various TSC-Diode systems and the server, a NATS server is established and configured to operate in a continuous listening mode. The goal of this server is to efficiently receive and process data transmitted by multiple diode systems.

Each diode, upon receiving the necessary data (including SPaT and CRC) from the TSC, publishes (i.e. the World side STM32 publishes) its base64 encoded data to a distinct subject format: "**traffic.<UNIQUE_ID>**". This approach ensures that data from different diode systems can be easily distinguished based on their unique identifiers (UNIQUE_ID). The Python script running on the NATS server is then designed to receive data streams from multiple diodes without the need for separate listening configurations for each diode. The script achieves this by subscribing to the general subject "**traffic.***", the NATS server can capture data streams from all diode systems in a straightforward manner, simplifying the data reception process.

Once the data arrives at the NATS server, the Python script performs a series of

operations to properly process and interpret the information. The script decodes the unique identifier from the subject to determine the originating intersection associated with the data. This unique identifier ensures that the server can correctly attribute the incoming data to the corresponding intersection.

4.4.2 Simulating messages on NATS server

The NATS client offers a versatile tool for testing and simulating various aspects of your system. By installing the NATS client on a Mac or Linux PC, you can effectively emulate both diode data transmission to the NATS server and the updates regarding position and light status to the intersection service. This enables you to replicate and assess different scenarios without the need for physical devices or deployments.

With the NATS client, you can generate data payloads that simulate the information a diode would transmit to the NATS server. This data can be published to relevant subjects, similar to how an actual diode would communicate. Furthermore, you can emulate position and light status updates that would typically be received from vehicles or devices near intersections.

By setting up this testing environment, you can evaluate how your system handles different data inputs, interactions, and potential challenges. This approach allows you to fine-tune and optimize your system's behavior, ensuring that it responds accurately and efficiently to varying conditions and inputs. Ultimately, using the NATS client for emulation aids in robust testing and validation before deploying the system in real-world scenarios.

For this the nats "**CLI**" tool must be installed. Also a local NATS server must be installed if the one has no access to a remote server. To install NATS CLI tool, in MacOS (Homebrew must be installed separately)

```
brew tap nats-io/nats-tools
brew install nats-io/nats-tools/nats
```

To install NATS server:

```
brew install nats-server
```

Further info in the installations can be found from "https://docs.nats.io/nats-concepts/what-is-nats/walkthrough_setup"

You have the flexibility to utilize the NATS demo server as part of your testing and development process. The demo server can be accessed through the NATS connection "nats://demo.nats.io" (which is not a standard browser URL, but rather a connection URL intended for NATS client applications This URL should be provided to the NATS client application to establish a connection).

For the project, a specific server was established at "Agricultural & Biological Engineering," Purdue University. The server domain is "***ibts-compute.ecn.purdue.edu***" with the corresponding IP address being **128.46.199.13**. The server operates on port**4223**. This server is maintained and managed by the OATSCenter (Open Ag Technologies and Systems Center).

This server setup allows one to interact with the NATS messaging system in a controlled environment, enabling to test various aspects of the system's communication, data transmission, and integration with the NATS infrastructure. In the following sections, one can learn more about how your system utilizes and interacts with this server to achieve the desired functionalities.

4.4.2.1 Emulating a diode If a server is already setup, then telnet to the server will help to establish a client connection with the server. If not a local server can be run to contnuously listen to the client. An isntance of client-server messaging setup on a MAC terminal is shown in Figure

If a server is already set up and running, one can establish a client connection to the server using the "telnet" command. The nats cli command-line tool allows to interact with remote servers using the telnet protocol. Telnet can be used to establish a simple client-server messaging setup, enabling to send and receive data between the client and server.

If a server is not already setup, one can set up a local server on the terminal that continuously listens for client connections. An instance of client-server messaging setup on a MAC terminal is shown in Figure 32

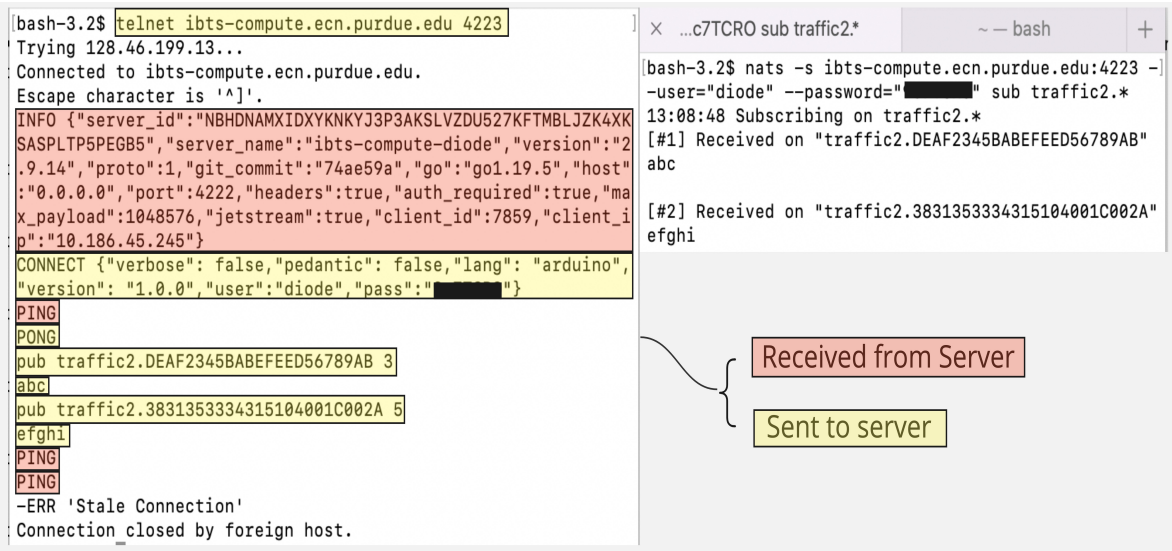

Figure 32: Emulating Diode messages with NATS CLI tool

Figure 33 shows sending messages from NATS CLI to demo.nats.io

Subscribing to the subject "**traffic.***" on the NATS server will allow one to confirm the flow of base64 encoded data, provided that the diode is actively broadcasting its data. By subscribing to this subject, one will be able to receive and observe the data packets being transmitted from the Data Diode system. This is shown in Figure 34.

4.4.2.2 Emulating the NATS script for the Web App Commands to emulate the position to intersection service and light status update, similar to the **"nats_parsing.py"**

```
bash-3.2$ telnet demo.nats.io 4222
Trying 147.75.47.215...
Connected to demo.nats.io.
Escape character is '^]'.
INFO {"server_id":"NC5BFXEYWVQUHQQMG7FVARRY3AYVOKP7H2FIUI2ZTH55Z5
FWHWJ2LOXY","server_name":"us-south-nats-demo","version":"2.9.21"
,"proto":1,"git_commit":"b2e7725","go":"go1.19.12","host":"0.0.0.
0","port":4222,"headers":true,"tls_available":true,"max_payload":
1048576,"jetstream":true,"client_id":299371,"client_ip":"128.210.
106.58","nonce":"CtYJK-Smw04ctPA"}
CONNECT {"verbose": false,"pedantic": false,"lang": "arduino","ve
rsion": "1.0.0"}
PING
PONG
pub traffic2.38313533343151040001C002A 5
abcde
```

Figure 33: Using demo.nats.io with NATS CLI tool

```
bash-3.2$ nats -s ibts-compute.ecn.purdue.edu:4223 --user="diode" --password="[redacted]" sub traffic.*
13:36:10 Subscribing on traffic.*
[#1] Received on "traffic.38313533343151160034003C"
zRABAEMAQwAAAAAAAAAAAAgAKAAoACgAKAAAAAAMAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAABQAAAAAAAAAAAAAAAAYAAAAAAAAAAAA
AAAAAAHAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAkAAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAAACwAAAAAAAAAAAAAAAAwAAAAAAA
AAAAAAAAANAAAAAAAAAAAAAAAADgAAAAAAAAAAAAAAAA8AAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAA//0AAAAC//0AAgAA//8AAAA
AAAAAACADEF4Av0oAAAAAAAPgGxj7

[#2] Received on "traffic.38313533343151160034003C"
zRABAEIAQgAAAAAAAAAAAAgAJAAkACQAJAAAAAAMAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAABQAAAAAAAAAAAAAAAAYAAAAAAAAAAAA
AAAAAAHAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAkAAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAAACwAAAAAAAAAAAAAAAAwAAAAAAA
AAAAAAAAANAAAAAAAAAAAAAAAADgAAAAAAAAAAAAAAAA8AAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAA//0AAAAC//0AAgAA//8AAAA
AAAAAACADEF8Av0oAAAAAAANxT+nG

[#3] Received on "traffic.38313533343151160034003C"
zRABAEEAQQAAAAAAAAAAAAgAIAAgACAAIAAAAAAMAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAABQAAAAAAAAAAAAAAAAYAAAAAAAAAAAA
AAAAAAHAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAkAAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAAACwAAAAAAAAAAAAAAAAwAAAAAAA
AAAAAAAAANAAAAAAAAAAAAAAAADgAAAAAAAAAAAAAAAA8AAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAA//0AAAAC//0AAgAA//8AAAA
AAAAAACADEGAAv0oAAAAAAANbYO7E

[#4] Received on "traffic.38313533343151160034003C"
zRABAEAAQAAAAAAAAAAAAAgAHAAcABwAHAAAAAAMAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAABQAAAAAAAAAAAAAAAAYAAAAAAAAAAAA
AAAAAAHAAAAAAAAAAAAAAAACAAAAAAAAAAAAAAAAAkAAAAAAAAAAAAAAAAKAAAAAAAAAAAAAAAACwAAAAAAAAAAAAAAAAwAAAAAAA
AAAAAAAAANAAAAAAAAAAAAAAAADgAAAAAAAAAAAAAAAA8AAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAA//0AAAAC//0AAgAA//8AAAA
AAAAAACADEGEAv0oAAAAAAAOzVumC
```

Figure 34: Subscribing to the subject **traffic.***

script to the web app is provided in README file in the ”mobile” section of the github repository. Figures 35 and 36 show the commands in action on the MAC terminal.

```
bash-3.2$ nats -s 128.46.199.13:4223 --user "diode" --password "[redacted]"
reply "light-nearest" '{"intersectionId":53694,"signalGroup":4}'
14:03:33 Listening on "light-nearest" in group "NATS-RPLY-22"
14:03:36 [#0] Received on subject "light-nearest":

{"lat":40.4294804,"lon":-86.9126953}
14:03:37 [#1] Received on subject "light-nearest":

{"lat":40.4294806,"lon":-86.9126958}
14:03:39 [#2] Received on subject "light-nearest":

{"lat":40.4294808,"lon":-86.9126963}
14:03:40 [#3] Received on subject "light-nearest":

{"lat":40.4294808,"lon":-86.9126963}
14:03:41 [#4] Received on subject "light-nearest":

{"lat":40.4294809,"lon":-86.9126963}
14:03:42 [#5] Received on subject "light-nearest":
```

Figure 35: Emulating ”**light-nearest**” reply

4.4.3 Decoding the base64-encoded data and parsing the fields of the NTCIP 1202v2 TSCBM SPaT data

Next the script decodes the base64 encoded data payload that it receives from the diode. This payload contains the SPaT information along with a CRC value for data integrity verification. The CRC is a 4-byte value that is used to verify the accuracy and integrity of the received data. The script calculates the CRC for the received data and compares it with the CRC value included in the payload. If the calculated CRC matches the received CRC, it indicates that the data hasn't been corrupted during transmission. Upon successful CRC verification, the script proceeds to parse the data in the payload and publish it to the intersection specific subject.

It's important to note that while a CRC check can help detect some types of data corruption, it's not foolproof. There's a possibility of false positives, where a corrupt message might still have a correct CRC. For higher levels of data protection, cryptographic hash functions like MD5 or SHA can be used. These hashes provide more robust integrity verification, although they are slower to compute compared to CRC32. Using such cryptographic hashes adds an extra layer of confidence in the data's accuracy and integrity.

Upon decoding the hexadecimal data, the script proceeds to parse it in accordance with the NTCIP Protocol. The SPaT (Signal Phase and Timing) data contains a collection of fields that provide comprehensive information about the timing of different phases at the intersection. These fields include measurements related to the durations of various traffic signal phases.

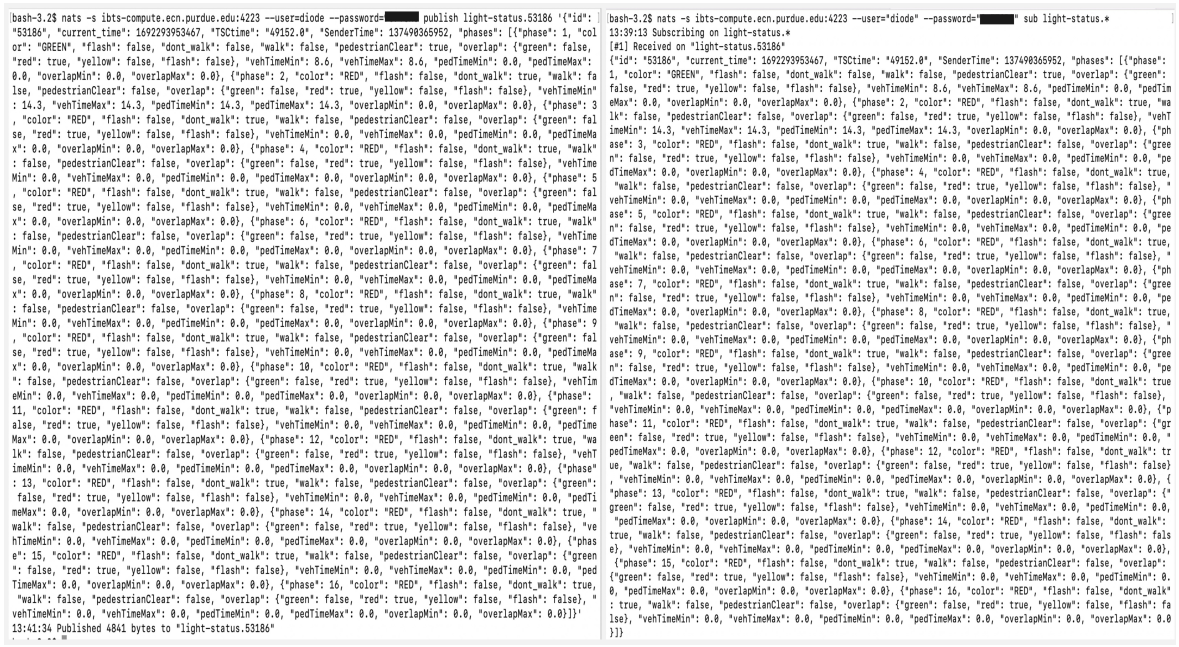

Figure 36: Emulating RED Light Status

Two parameters "vehTimeMin" and "vehTimeMax" in each phase is of particular importance in predicting the light status of the current lane. The parameters exhibit continuous variations based on the prevailing traffic conditions. The light status of a traffic lane changes when these two values become equal and then both decrease simultaneously. In specific scenarios, when both "vehTimeMin" and "vehTimeMax" are equal and assume a value "x," this indicates that the light status of a specific traffic phase is expected to change within "x" milliseconds.

This equalization of "vehTimeMin" and "vehTimeMax" serves as a countdown timer that offers valuable information to users. By providing an advance notice of the imminent change in traffic light status, users can anticipate the upcoming shift and make appropriate adjustments to their driving behavior. Additionally, this piece of information, combined with the measured latency of the system, contributes to predicting future light statuses with a higher level of accuracy. In essence, this mechanism enhances road safety and optimizes traffic flow by allowing drivers and other stakeholders to anticipate and adapt to changes in traffic signal timings more effectively.

For a more detailed understanding of the TSCBM structure, you can refer to Figure 37 in the documentation. Further information about SPaT and intersection phase timing can be obtained from resources such as the "*Econolite Connected Vehicle SPaT and Cabinet Guide*" and the "*V2I Hub Interface Control Document (ICD)*." These documents are available in the manuals section of the GitHub repository associated with the project. By studying these documents, you can gain insights into the specifics of SPaT data and how intersection timing details are encoded and transmitted through the diode system.

Traffic Signal Controller Broadcast Message

Byte-Map Structure of the Broadcast Message, Version #2.

Bytes	Description	
Byte 0:	DynObj13 response byte (0xcd)	
Byte 1:	number of phase/overlap blocks below (16)	
Bytes 2-14:	0x01 (phase#)	(1 byte)
	VehMinTimeToChange.1	(2 bytes)
	VehMaxTimeToChange.1	(2 bytes)
	PedMinTimeToChange.1	(2 bytes)
	PedMaxTimeToChange.1	(2 bytes)
	OvlpMinTimeToChange.1	(2 bytes)
	OvlpMaxTimeToChange.1	(2 bytes)
< repeat for each phase and overlap – bytes 15-196 >		
Bytes 197-209:	0x10 (phase#)	(1 byte)
	VehMinTimeToChange.16	(2 bytes)
	VehMaxTimeToChange.16	(2 bytes)
	PedMinTimeToChange.16	(2 bytes)
	PedMaxTimeToChange .16	(2 bytes)
	OvlpMinTimeToChange .16	(2 bytes)
	OvlpMaxTimeToChange .16	(2 bytes)
Bytes 210-215:	PhaseStatusReds	(2 bytes bit-mapped for phases 1-16)
	PhaseStatusYellows	(2 bytes bit-mapped for phases 1-16)
	PhaseStatusGreens	(2 bytes bit-mapped for phases 1-16)
Bytes 216-221:	PhaseStatusDontWalks	(2 bytes bit-mapped for phases 1-16)
	PhaseStatusPedClears	(2 bytes bit-mapped for phases 1-16)
	PhaseStatusWalks	(2 bytes bit-mapped for phases 1-16)
Bytes 222-227:	OverlapStatusReds	(2 bytes bit-mapped for overlaps 1-16)
	OverlapStatusYellows	(2 bytes bit-mapped for overlaps 1-16)
	OverlapStatusGreens	(2 bytes bit-mapped for overlaps 1-16)
Bytes 228-229:	FlashingOutputPhaseStatus	(2 bytes bit-mapped for phases 1-16)
Bytes 230-231:	FlashingOutputOverlapStatus	(2 bytes bit-mapped for overlaps 1-16)
Byte 232:	IntersectionStatus (1 byte)	(bit-coded byte)
Byte 233:	TimebaseAscActionStatus	(1 byte) (current action plan)
Byte 234:	DiscontinuousChangeFlag	(1 byte) (upper 5 bits are msg version #2, 0b00010XXX)
Byte 235:	MessageSequenceCounter	(1 byte) (lower byte of up-time deciseconds)
Byte 236-238:	SystemSeconds (3 byte)	(sys-clock seconds in day 0-84600)
Byte 239-240:	SystemMilliSeconds (2 byte)	(sys-clock milliseconds 0-999)
Byte 241-242:	PedestrianDirectCallStatus	(2 byte) (bit-mapped phases 1-16)
Byte 243-244:	PedestrianLatchedCallStatus	(2 byte) (bit-mapped phases 1-16)

Figure 37: Traffic Signal Controller Broadcast Message Version #2 Byte-Map Structure

4.4.4 Associating the UNIQUE ID with the corresponding intersection and re-publishing the parsed data to the relevant intersection subject

Publishing the parsed data to the appropriate intersection subject involves a process where the unique identifier (UNIQUE ID) assigned to each diode plays a crucial role. The diode itself doesn't need to possess knowledge about the specific intersection it corresponds to. Instead, the responsibility of mapping this UNIQUE ID to the actual intersection number is assigned to the python script running on the NATS server.

To achieve this mapping, the python script maintains a record of each diode's UNIQUE ID and its associated intersection number. This mapping can be established through various means instead of creating the mapping in script. For instance, the mapping information could be stored in a PostgreSQL database, where the UNIQUE ID serves as the primary key and is linked to the corresponding intersection number.

Upon receiving data from a diode, the python script decodes the UNIQUE ID embedded within the incoming data. With this decoded UNIQUE ID, the script can instantly identify the relevant intersection number based on the established mapping between the Diode ID and the intersection number in the script. This intersection number is then used as a reference point to ensure that the parsed data is directed to the appropriate intersection subject for further processing.

To better explain the above procedure, NATS script listens on the subject "**traffic.***" and receives the data as a string of characters that encode the original SPaT data. While the UNIQUE ID of the diode is obtained from the sunject name, to obtain the original binary data, a base64 decoder is applied to convert the encoded string back to its SPaT representation. Post CRC validation, the data is parsed as per NTCIP protocol. The parsed data is converted into a structured JSON (JavaScript Object Notation) format. JSON is a widely used data interchange format that allows data to be represented in a human-readable and machine-readable format. The parsed SPaT information is organized into JSON fields, making it easier to work with and transmit.

The previously mapped intersection ID (obtained from the unique ID) is used as part of the new subject, "light_status.<intersection_id>". The JSON data unit, containing the decoded SPaT information, is published to this specific subject. By doing so, the data is made available for subscribers who are interested in monitoring the traffic light status of a particular intersection.

Applications or systems interested in accessing the traffic light status of a specific intersection can subscribe to the "light_status.<intersection_id>" subject. Once subscribed, they will receive the published JSON data unit whenever a new update is available (every 100ms). This process allows subscribers to obtain real-time traffic signal information for their intended intersection of interest. This well-structured approach ensures that traffic signal information is effectively transmitted, received, and utilized for monitoring and analysis purposes. A re-published SPaT JSON message unit is shown in Figure 38.

4.4.4.1 MAP Data Unit The MAP Data Unit is a critical component within the project, providing a mathematical model of intersections based on the SAE J2735 standard's data elements and definitions. The SAE J2735 standard defines a comprehensive

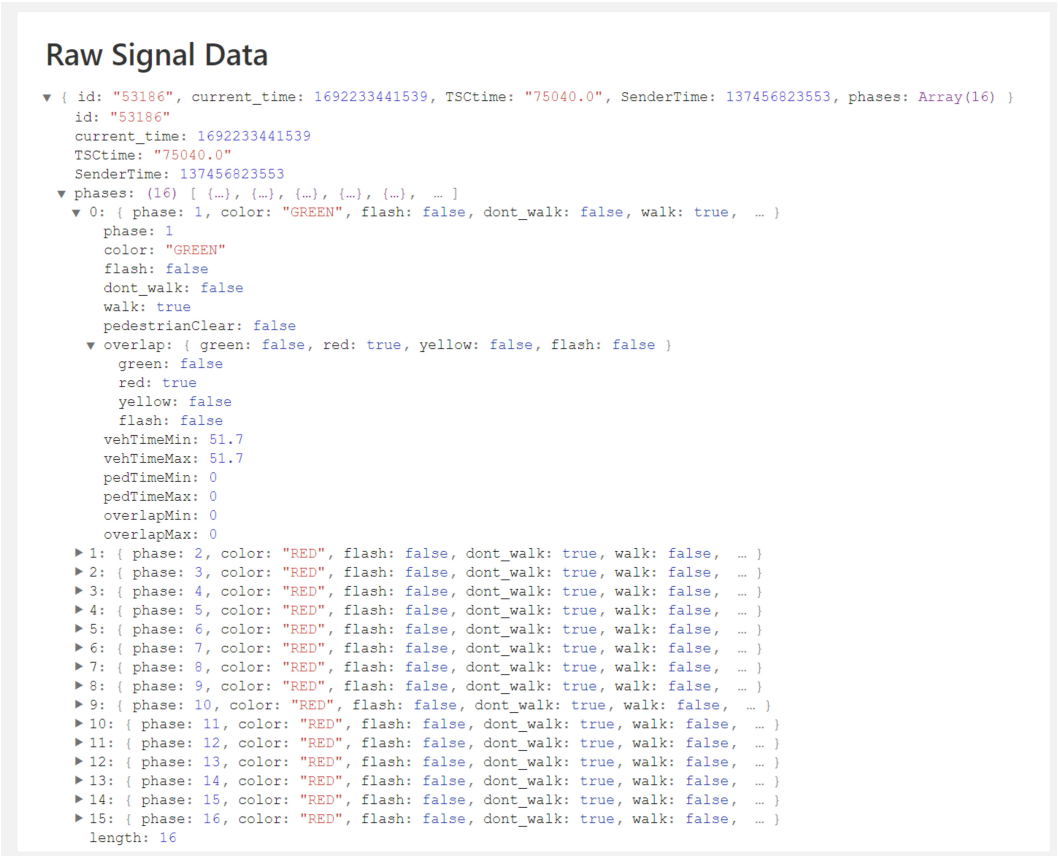

Figure 38: Re-published SPaT JSON Message Unit

set of messages, data frames, and data elements that are utilized by applications making use of vehicle-to-everything (V2X) communication systems. For this project, the focus is on specific aspects of the MAP data unit to achieve our goals, although the data unit encompasses a broader range of features that could potentially be leveraged in the future.

Essentially, the MAP data unit serves as a means to describe intersections in a standardized way, making it possible for vehicles and infrastructure components to communicate effectively and share essential information. By adhering to the SAE J2735 standard, the project ensures that the data exchanged between different elements of the transportation system can be seamlessly understood and utilized.

To facilitate the creation of the MAP data unit, a specialized tool was employed. This tool can be accessed at the https://webapp.connectedvcs.com/ called the Intersection Situation Data (ISD) Message Creator. This tool allows users to configure and generate MAP data units that accurately represent intersections according to the SAE J2735 standard. The resulting MAP data unit provides the foundational information needed for the accurate interpretation of traffic scenarios, enabling effective data processing, analysis, and decision-making in the context of the project's objectives.

Creating a MAP data unit involves a structured procedure that helps accurately define the characteristics and movements of vehicles within an intersection. The process ensures that the resulting MAP data unit adheres to the SAE J2735 standard and contains the necessary information for effective communication and analysis. Here is the step-by-step procedure for creating a MAP data unit:

1. **Identify Ingress Lanes :** Start by identifying all lanes that approach the intersection. These are known as ingress lanes. It's important to number these lanes starting from the Northeast most lane and proceeding in a systematic order.

2. **Define Intersection Names :** The intersection names should be formatted as "<Major Street>and <Minor Street>". This naming convention helps clearly identify the streets involved in the intersection.

3. **Understand Allowed Movements :** Determine the movements that are allowed in each ingress lane. This includes understanding which directions of travel are permitted for vehicles in each lane, such as straight pass, left turn, right turn, etc.

4. **Assign Reference Numbers to Ingress Lanes :** Assign unique reference numbers to each ingress lane. These reference numbers play a crucial role in identifying and differentiating the lanes within the MAP data unit.

5. **Determine Signal Group Assignments :** Identify which signal group is responsible for controlling each lane. Signal groups are responsible for regulating the traffic movements within the intersection, and associating lanes with signal groups is essential for accurate communication.

6. **Record Geographic Coordinates :** For each lane, record the latitude and longitude coordinates for key points. These points include the intersection center (reference point), as well as the start and stop points of each lane. These

coordinates help precisely define the spatial layout of the intersection and its lanes.

By following this procedure, one can systematically gather and organize the necessary information to construct a comprehensive MAP data unit. This unit will accurately represent the intersection, its lanes, allowed movements, signal group assignments, and geographic coordinates. The resulting MAP data unit will serve as a standardized representation that can be used for communication, analysis, and decision-making within the project's scope.

Owosso, Michigan, was chosen as the location for implementing the data diode system. The implementation of the data diode in the city of Owosso involved a systematic approach that required the creation of a MAP data unit to accurately represent the intersections in the area. Within the city, a total of 21 intersections were identified for the project's scope. Out of these intersections, the diode system was practically deployed in 2 specific intersections (further details on these intersections will be provided later).

Details such as ingress lanes, allowed movements, signal group assignments, and geographic coordinates were accurately captured for all 21 intersections and 21 MAP files were created.Each of the 21 intersections in Owosso was assigned a unique reference number. These reference numbers served as identifiers for each intersection and helped DOT personnel and project stakeholders clearly identify and reference specific intersections. For example, an intersection reference number like "53186" would correspond to the intersection of "Washington Main St & N Washington St."

The resulting MAP data unit provided a standardized representation of each intersection's characteristics, movements, signal group assignments, and spatial layout. This unit served as a critical component for accurate communication and data processing within the data diode system.

The 12-byte unique Diode ID from the traffic signal controller (TSC) cabinet at a particular intersection was associated with the corresponding intersection reference number in the NATS script. This mapping allowed the system to identify and route data from the diode to the appropriate intersection subject.

The task of creating the MAP data unit was undertaken by David Hong (davidhdw@vt.edu) and Dr. Montasir Abbas (abbas@vt.edu) from the Virginia Tech civil engineering department. They meticulously gathered and organized the intersection information using the ISD Message Creator tool and constructed the MAP data unit for each intersection. A snippet from the ISD Message Creator tool used for creating MAP data unit is shown in Figure 39

The MAP files containing information about the 21 intersections in Owosso are available on the project's GitHub page. These files can be accessed in the "MAP_files" folder, providing a comprehensive overview of each intersection's specifications.

Figure 39: ISD Message Creator Tool Sample View

4.4.5 Responding to queries from the app by providing information about the nearest intersection number, current traffic light status, and allowed maneuvers in the app's queried location

PostgreSQL table was created and was utilized for querying and retrieving relevant information from the MAP data unit. The table is created to store information extracted from the MAP data unit. This table contains columns for intersection number, lane ID, signal group, maneuvers, latitude and longitude of lane nodes. This format allows for efficient storage and retrieval of lane-specific information.

To extract the relevant information from the MAP File and convert it to a Postgres uploadable table a Python script named **"map_to_postgress.py"** is used. This script reads the lane information and creates an output file ("owosso_json.txt" and "owosso_postgress.csv") that contains the formatted data. This data can be directly imported into the PostgreSQL database table called "***lanes***".

By default, the data is queried into the public schema. However, if a separate schema is desired, adjustments to SQL queries are necessary. For instance, the SQL query in the "nats_parsing.py" script can be modified to include the desired schema for querying. Specifically,

```
SELECT intersection_id, signal_group, maneuvers,
   ST_DistanceSphere(geo, ST_SetSRID(ST_MakePoint(' + str(
   curr_location["lat"]) + ',' + str(curr_location["lon"])
   +'), 4326)) as dist FROM postgres.natsql.lanes ORDER BY
   dist ASC LIMIT 1')
```

needs to be modified.

When new GPS coordinates are obtained, a query is made to find the nearest lane using spatial functions provided by PostgreSQL. The query calculates the distance between the queried GPS coordinates and the coordinates of lane nodes stored in the table. The "ST_DistanceSphere" function is used for this purpose. The queried results provide information about the nearest lane, including its signal group, allowed

maneuvers, and intersection number. This data can be used to determine the traffic signal status and relevant information for that specific location.

The ”map_to_postgress.py” script essentially acts as a bridge between the extracted MAP data and the PostgreSQL database. It enables the efficient and concise storage of the MAP data units of each intersection. The cmd output of this ”map_to_postgress.py” script is shown in the Figure 40.

```
C:\Users\kmani\Documents\DATA_DIODE_FINAL>python map_to_postgress.py
{
  "12544": {
    "reference point": "(43.004933200345775,-84.17680444389076)",
    "1": {
      "laneManeuvers": [
        "right",
        "straight"
      ],
      "signal_group": 2,
      "line": "((43.005060834574145,-84.17690054463758),(43.00540948699405,-84.17689853298177))"
    },
    "2": {
      "laneManeuvers": [
        "right"
      ],
      "signal_group": 2,
      "line": "((43.005060834574145,-84.17685025321997),(43.00541095809707,-84.17684824156326))"
    },
    "3": {
      "laneManeuvers": [
        "left"
      ],
      "signal_group": 5,
      "line": "((43.005060834574145,-84.17681404339838),(43.00541095809707,-84.17681404339838))"
```

Figure 40: ”map_to_postgress.py” script output

The ”***lanes***” table created in Postgres is shown in Figure 41.

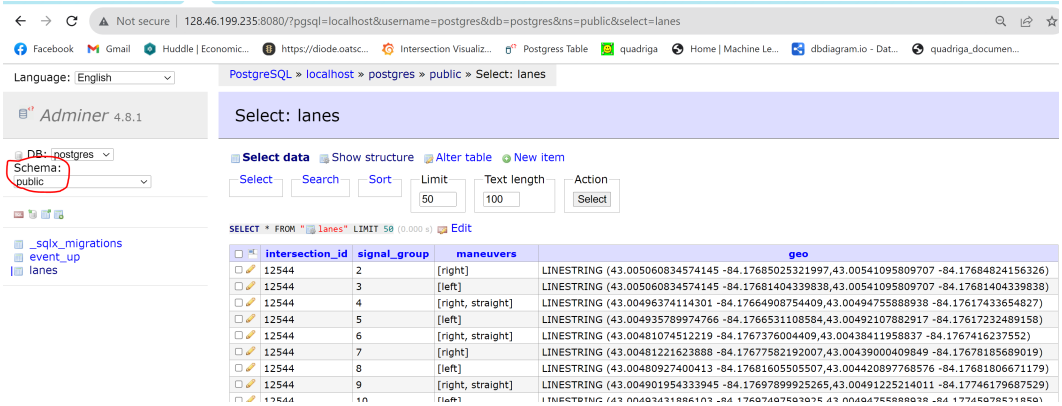

Figure 41: *lanes* table in Postgres

The mobile app subscribes to a NATS subject called ”light-nearest” by providing its current GPS coordinates. This subject is used to initiate the process of retrieving relevant traffic information for the user’s location. The same NATS script ”nats_parsing.py” is also set up to listen to the ”light-nearest” subject. Upon receiving a subscription request from the mobile app, this script retrieves the GPS coordinates provided by the app.

The script then queries the PostgreSQL database using the provided GPS coordinates. The spatial functions of PostgreSQL, such as "ST_DistanceSphere," are used to calculate the distance between the queried GPS coordinates and the coordinates of lane nodes stored in the database.

The result of the query includes information about the nearest lane. This information includes details like the lane's signal group, allowed maneuvers, and intersection ID. This data is retrieved from the database based on the calculated proximity to the queried GPS coordinates.

Once the lane information is obtained from the database, the script sends a response back to the mobile app. This response includes details about the nearest lane, the corresponding signal group, maneuvers allowed in that lane, and the intersection ID of the queried GPS coordinate.

Armed with the intersection ID obtained from the previous step, the mobile app now knows which intersection it needs to monitor. It subscribes to a NATS subject named "light-status.<intersection_id>." This subject corresponds to the parsed JSON data for the specified intersection's traffic signal status. As traffic information updates are published to the "light-status.<intersection_id>" subject, the mobile app receives these updates. The app's user interface displays the real-time traffic information obtained from the "light-status.<intersection_id>" subject. Users can reason the current traffic signal status, predicted light changes, and any other relevant information based on their location.

The file "***nats_parsing.py***" can be found in the git hub code The output of "***nats_parsing.py***" is shown in the Figure 42. These serial prints can be disabled as well.

A pictorial overview of the data routing mechanism from the diode(s) to the intersection specific subject explained so far is shown in Figure 43. The **Backend Script** in this figure is "***nats_parsing.py***".

4.5 Web App Side of the System

Svelte tool was used for building a simple web application. Svelte is a modern web application framework that offers a unique approach to building user interfaces. Unlike other frameworks where the user interface components are interpreted at runtime, Svelte takes a different approach by compiling components into efficient JavaScript modules during build time. This compilation process optimizes the performance of the application and eliminates much of the overhead that traditional UI frameworks introduce.

Since Svelte compiles components into efficient JavaScript code, there is significantly less runtime overhead compared to other frameworks. This leads to faster load times and smoother user experiences, as unnecessary abstractions and runtime computations are minimized. It's compilation process results in smaller bundle sizes for your application. The generated JavaScript modules are tailored to the specific features used in your components, reducing the need for shipping unused code to the client's browser. It provides a rich library of components that you can use as a foundation for building

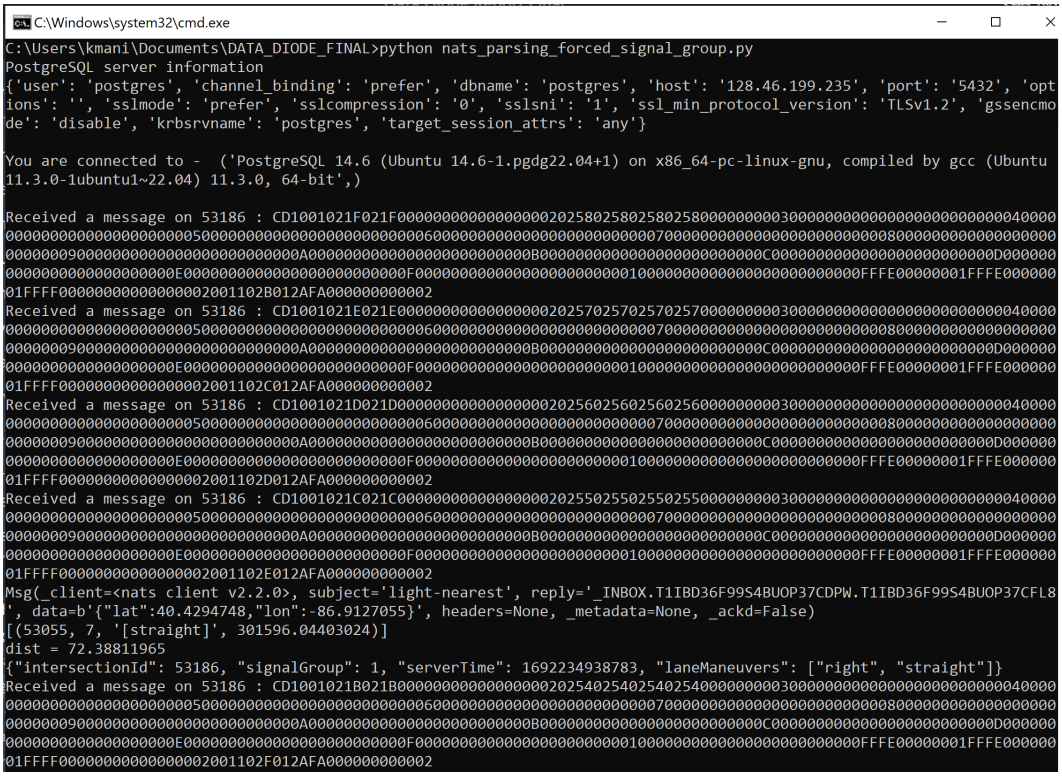

Figure 42: *"nats_parsing.py"* script output

your app. These components cover various aspects of UI, including forms, navigation, layout, and more. More info can be found from https://kit.svelte.dev/

4.5.1 Designing a user-friendly mobile application capable of capturing the user's GPS location

watchPosition() method of **navigator.geolocation** plugin is used to register a handler function that will be called automatically each time the position of the device changes. Since there was minimal attention to build the app as it was not our job, attention was not given to accuracy of the gps or the refresh rate of the gps. The direction info from the gps can also be used to obtain the correct lane the device (vehicle) is moving towards.

When the **watchPosition()** method is called, a callback function is provided that will be executed each time the device's position changes. This callback function can receive a Position object as an argument, which contains information about the device's geographical coordinates, timestamp of the reading, and more.

Once the callback function is registered, the browser will automatically invoke it whenever there is a change in the device's position. This can occur due to factors such as the device moving, changing its location, or if it receives more accurate GPS data.

The accuracy of the GPS readings and the refresh rate (how often the position is updated) depend on various factors, including the device's hardware, the browser's implementation, and the user's location. While using **watchPosition()**, the accuracy and refresh rate can impact the overall performance and usability of the application.

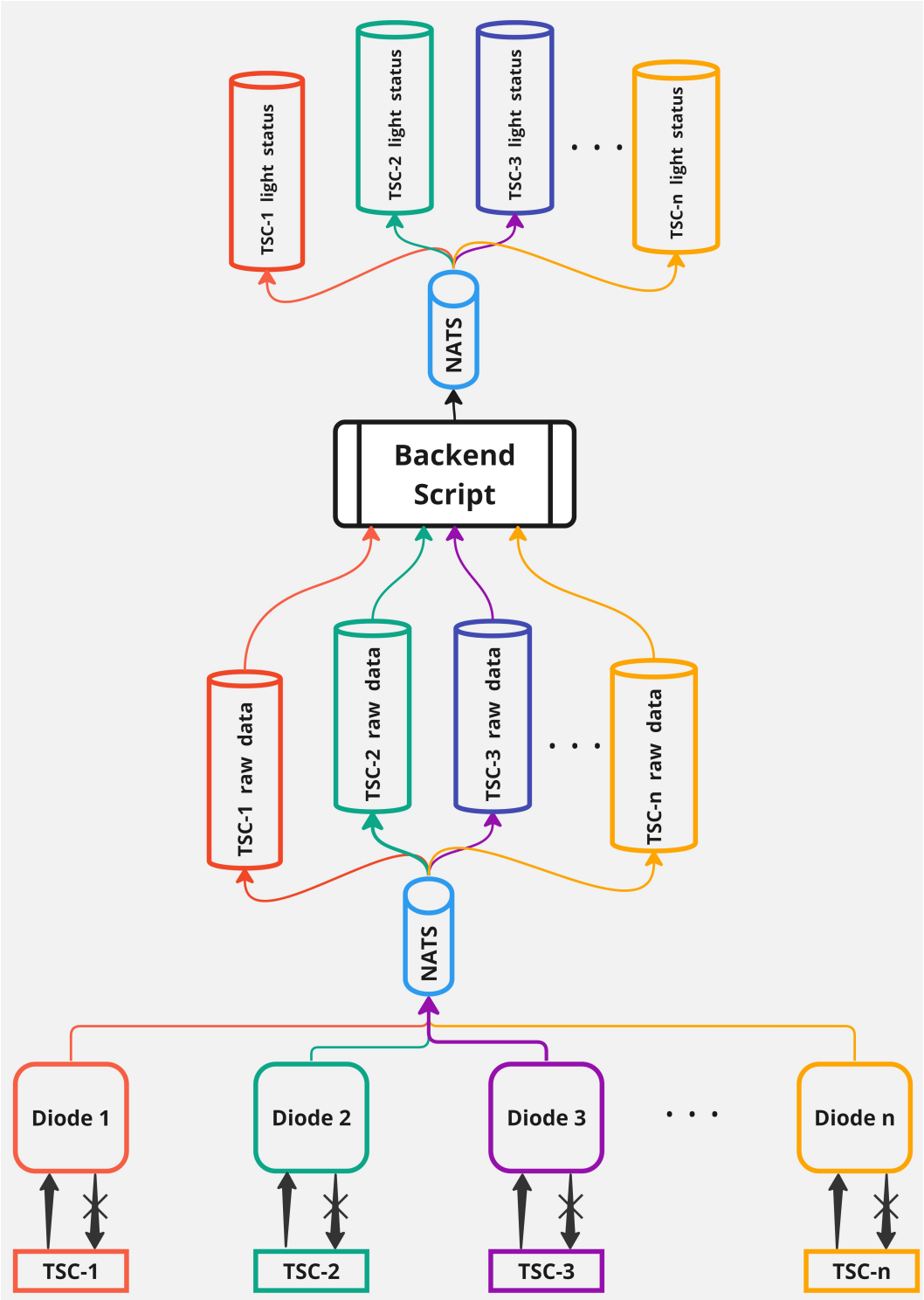

Figure 43: NATS Data Routing Mechanism

Higher accuracy and more frequent updates may result in better location tracking but could also consume more battery and data.

The GPS data provided by the **watchPosition()** method can include direction information, often referred to as "heading." This indicates the direction in which the device is moving. By analyzing this heading information, one can better determine the lane the device is likely traveling in. This feature is not implemented to keep the operation simple.

It's important to be aware that its accuracy can vary, and the app's behavior may differ across different devices and environments. Factors like signal strength, interference, and urban canyons can affect GPS accuracy.

4.5.2 Querying the server for the nearest intersection number based on the user's GPS coordinates

When the position changes, the app queries the "light-nearest" subject with the current GPS coordinates. This query allows the app to retrieve information about the nearest lane and intersection to the device's current location.

Using the obtained lane and intersection data, the app displays details such as lane maneuvers and the signal group associated with the lane. This information offers insights into the specific actions allowed within the lane, as well as the current signal group controlling that lane's traffic light.

It's important to note that the accuracy of the displayed information is dependent on the accuracy of the GPS coordinates received from the device. Variability in GPS accuracy can lead to variations in the precision of lane and intersection information displayed by the app

4.5.3 Subscribing to the NATS topic specific to the intersection number to retrieve the parsed SPaT message

The application leverages the intersection ID acquired in the previous step to establish a subscription to the "light-status.<intersection_id>" subject. This subscription enables the app to receive the JSON data unit that was parsed and published by the script for the corresponding intersection. The JSON data unit contains information about the status of traffic lights and other relevant details for that intersection.

As the device continues to move, the app continuously displays the received JSON data, updating the information to reflect the current state of traffic lights and related data. This dynamic display ensures that users receive real-time updates about the traffic conditions as they approach different intersections along their route.

4.5.4 Calculating and displaying latency information through a dedicated script

In the context of designing a system that relies on data transmission and real-time predictions, it's essential to consider the latency introduced by the communication process. In the reference to the current project, the latency can affect the accuracy of predicting the future light state at an intersection based on the received data. The

latency measurement process involved using the modified version of "tsc_simulator" script, which was employed to calculate the latency values. To capture this latency, the current time of the system, measured in milliseconds since the epoch, was embedded within one of the data fields and transmitted through the diode.

On the server side, the NATS script incorporated the reception time of the data into another field. When this data packet was received by the app, the Svelte app utilized the current time to calculate both the uplink latency and the end-to-end latency. This calculation involved comparing the time of transmission with the time of reception to determine the time taken for the data to traverse the system.

To obtain accurate latency measurements, the process was repeated for about 1000 packets, and the latencies from these packets were averaged to derive the average latency. Additionally, the minimum and maximum latencies among the 1000 odd packets were recorded to provide insights into the variability of latency within the system.

For conducting these latency measurements, the script "**latency_measurement.py**" was executed following similar procedures as the "tsc_simulator" script. To visualize the results, the modified version of the deployed app, named "mobile_latency," was utilized. Running the "**latency_measurement.py**" script and starting the app using "*npm run dev*" in the Visual Studio terminal allowed for the observation of the latency measurements on the local host link.

The NATS server is situated within the Agricultural & Biological Engineering department at Purdue University. The latency measurements were conducted with the app located in the Materials and Electrical Engineering department of Purdue University is :

Average End-to-End Latency(ms)190.4654594232059
Average Uplink Latency(ms):179.36016096579476
MAX End-to-End Latency(ms):419
MIN End-to-End Latency(ms):161
MAX Uplink Latency(ms):388
MIN Uplink Latency(ms):157
Number of packets : 1491

The Latency measured at Lansing signal shop, Michigan is :

Average End-to-End Latency(ms) : 309.8716075156576
Average Uplink Latency(ms): 218.58977035490605
MAX End-to-End Latency(ms): 733
MIN End-to-End Latency(ms): 258
MAX Uplink Latency(ms): 554
MIN Uplink Latency(ms): 190
Number of packets : 958

The Latency measured at (M52 x King St.), Owosso Roughly, 43°00'17.8"N 84°10'36.6"W is :

Average End-to-End Latency(ms) : 470.2997987927565
Average Uplink Latency(ms): 344.66901408450707
MAX End-to-End Latency(ms): 948
MIN End-to-End Latency(ms): 13
MAX Uplink Latency(ms): 833
MIN Uplink Latency(ms): 237
Number of packets : 994

As expected the values are larger than that at Purdue due to proximity of the server to Purdue location.

A well-designed system would take into account the average latency experienced during data transmission. This means estimating how long it takes for data to travel from the sender (e.g., the diode system) to the receiver (e.g., the app), including any processing time at intermediary points such as the NATS server.

In practice, latency can vary due to network conditions, processing times, and other factors. While aiming for minimal latency is ideal, it's realistic to expect some level of variance in latency measurements. This variance, often seen in the order of tens of milliseconds, is the error of interest. To mitigate the impact of latency on predicting the future light state, developers would typically implement mechanisms to account for this variability.

By considering the average latency and factoring it into calculations, the system can provide more accurate predictions of light changes at intersections. This adjustment helps compensate for the potential delays in data transmission, resulting in a more reliable and effective real-time information system.

The web app is deployed on https://diode.oatscenter.org/. The back end workflow of the designed app is shown in Figure 44.

The front end view of the app is shown in Figure 45.

4.6 Setting up Python Server to Listen to Modem Data in UDP version of the code

The initial version of the code did not include integration with the NATS messaging system. Instead, it focused on a straightforward UDP packet transmission to a remote server. In this version, the data from the Data Diode system was sent over a UDP connection to a designated server without the additional functionality and features provided by the NATS protocol. In this version, the data packet was constructed by directly appending the UNIQUE ID to the SPaT data, without any CRC or base64 encoding applied. Consequently, the total size of the data packet in this configuration was determined by the sum of the size of the SPaT data and an additional 12 bytes for the UNIQUE ID. This original implementation aimed to establish a basic data transmission mechanism for the project's requirements. For this realisation, **"receive_data.py"** in the git repository of section 3 realises a python program that connects to a public IP, binds to a UDP socket and continuously listens to the incoming traffic. The code can be run the while the user intends to operate the Data Diode to receive the TSC SPaT

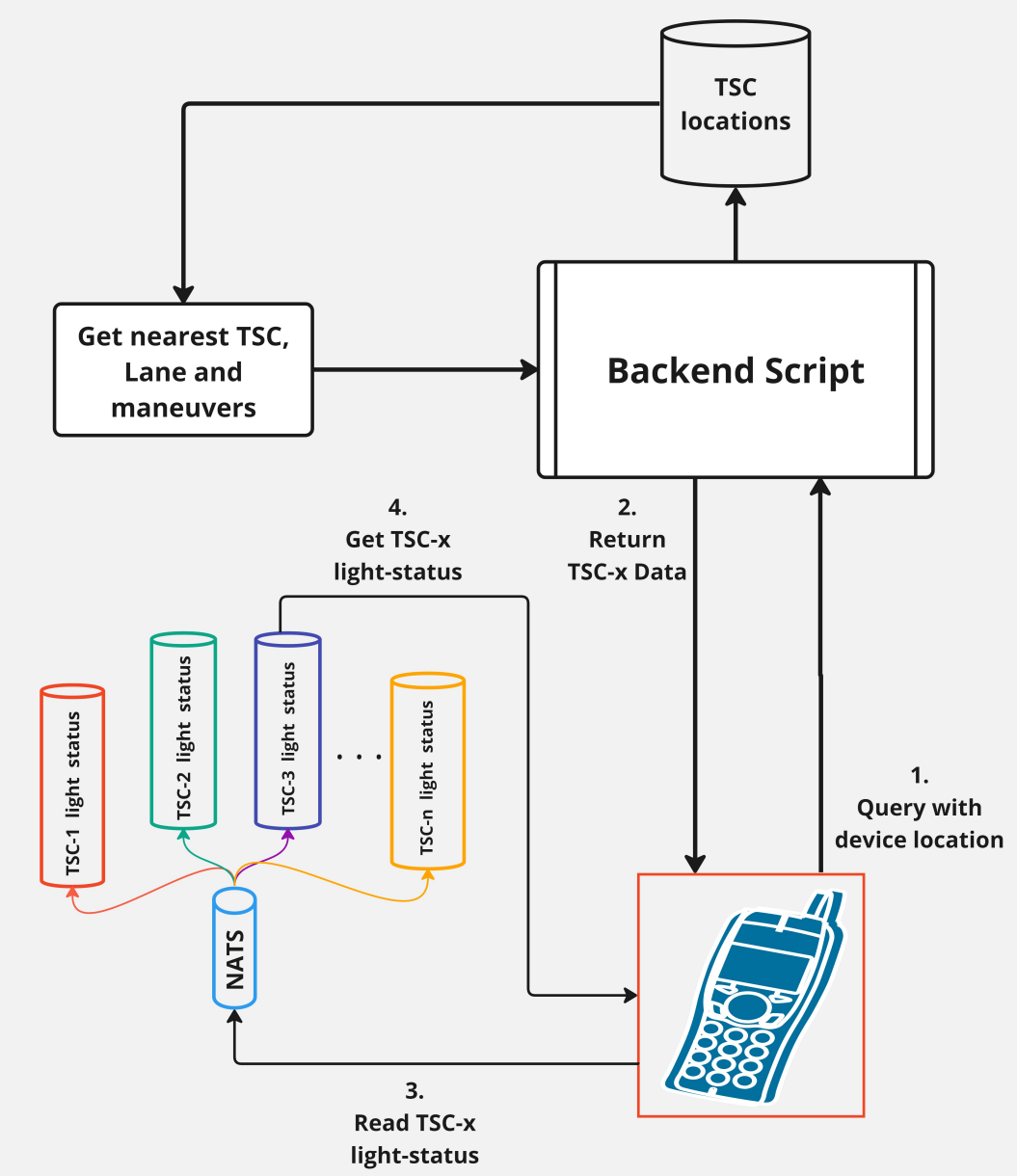

Figure 44: Web-App Back End Workflow

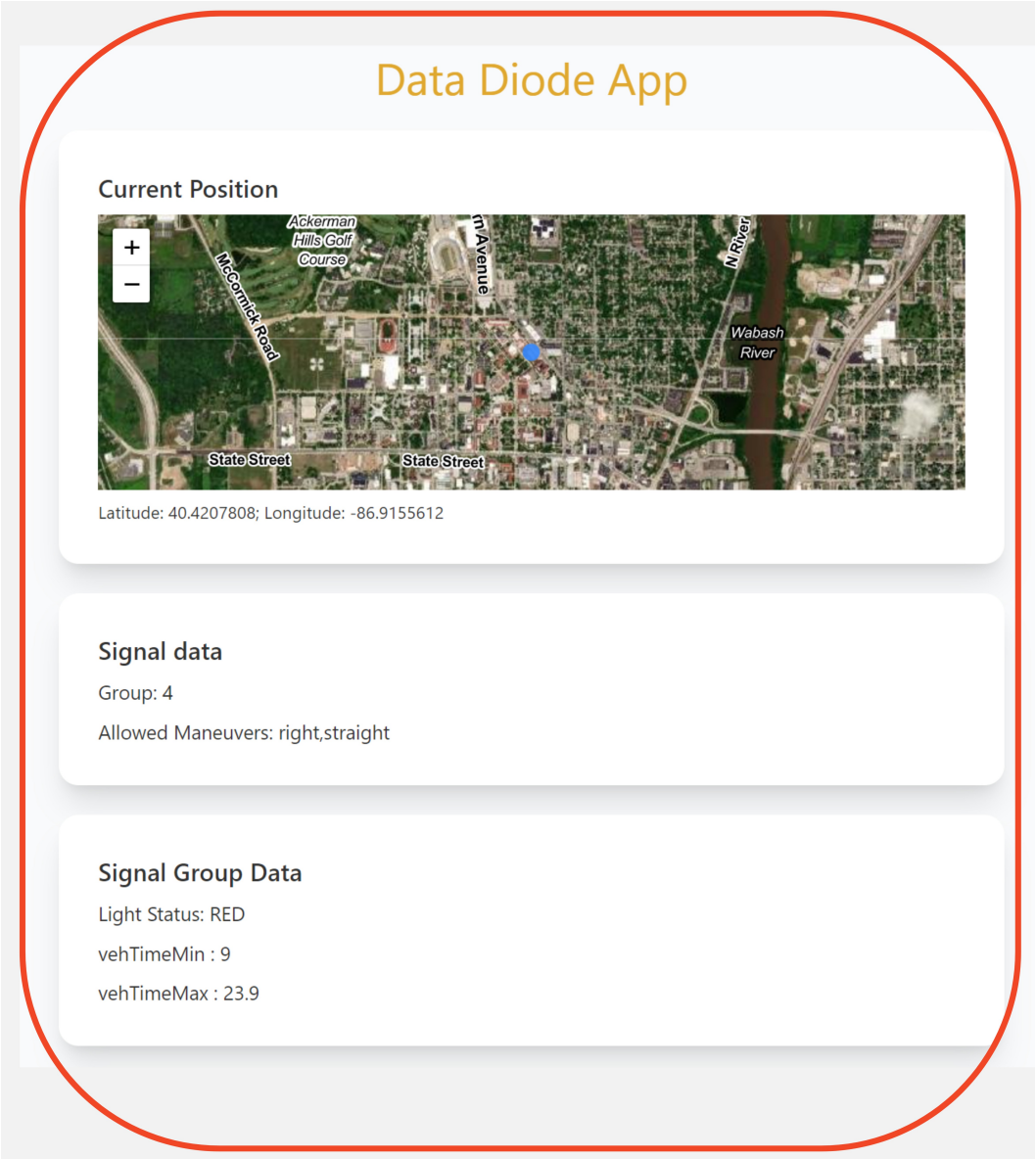

Figure 45: Web-App Front End View

messages.

The user needs to ensure availability of a public ipv4 address to which the PC running the ”receive_data.py” has access to. Alternatively, in case of non availability of a public IPv4 address, Amazon Web Services (AWS), DigitalOcean or other cloud services that provide a remote Virtual Machine with a Public IP enabled at users' disposal. Realising this on AWS is explained in detail in 4.7.

When the ”receive_data.py” is run, the received data(if any) is output to the console. A sample view of the console output is shown in Figure 46

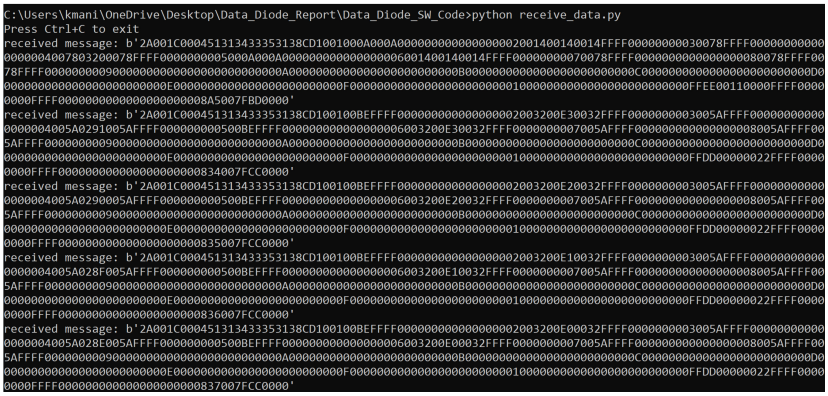

Figure 46: Python based Server's Console Output

4.6.1 Data Logging on to a Text File

The same ”receive_data.py” described in 4.6 logs the received data on to a text file - ”received_data.txt” in the same folder which the ”receive_data.py” is run. This text file is expected to contain the (12 byte unique ID + SPaT message) msg sent by the Controller side of the diode, logged every 100 ms.

Provided enough USB ports on it, all the three data loggings, i,e. the cell modem text file logging and the two STM32 serial interface debug loggings can be realised on the same PC.

4.7 Running the server on Amazon EC2 Instance

As briefly discussed in 4.6.1 Amazon EC2 Instance provides a public IP which can be used to route data traffic to from the cell modem in case of non availability of a dedicated ipv4 address for a user. Amazon Clouds use Software-Defined Networking (SDN) to manage things and a created instance(a virtual machine) will have a AWS' cloud rewriting the packets destined for the public IP to the private IP at the network's edge.

To practically realise the above Public-Pvt interconnect on the an Amazon EC2 instance, the following steps need to be followed.

1. SSH to an EC2 instance from your local system
2. The EC2 instance will have a Public IPv4 and a Private IPv4 address associated with it

3. In the Security tab of Instance summary window, click on the Security Group ID link and then click on "Edit inbound rules" option
4. Create a new rule of type Custom UDP with a specific port and IP address in the pvt address space. Alternatively one could set as 0.0.0.0 and listen to all allowed IP addresses.
5. In the EC2 linux terminal run "receive_data.py" described in 4.2.4 with port number and interface address same as that set in step 4
6. The "ServerIP" in the World side of the Diode needs to be set to as Public IPv4 from step 2 and port number same as that in step 4

   Two windows of the above process is shown in Figure 47

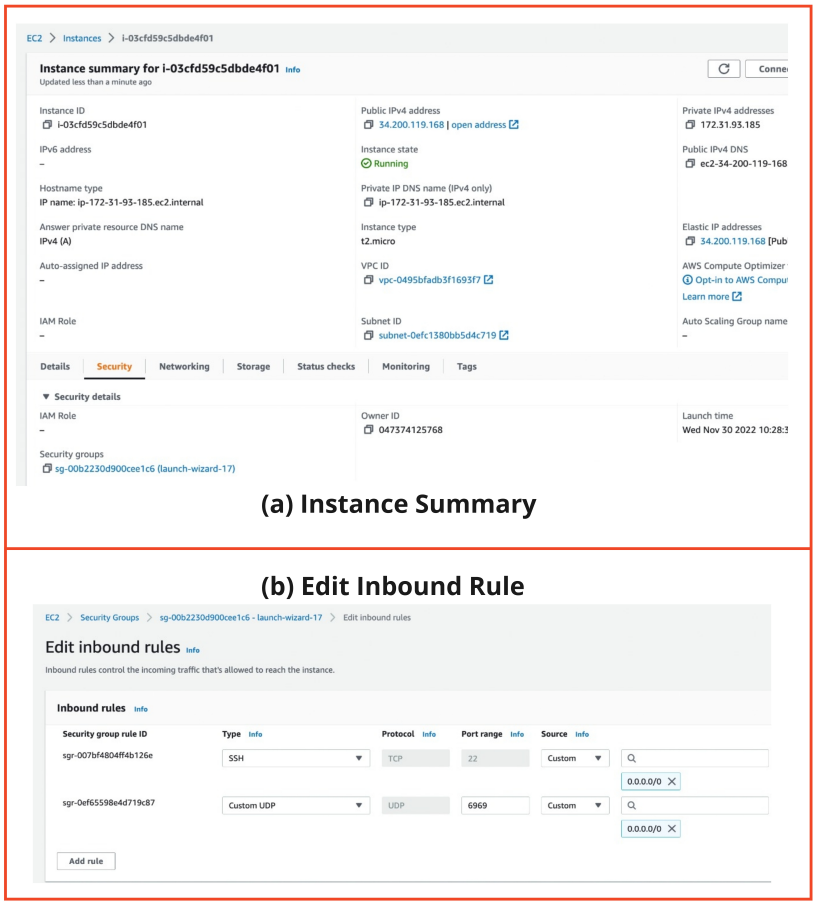

Figure 47: Creating Inbound Rule on an Amazon EC2 Instance

4.8 Grafana Dashboard

A Grafana dashboard was developed to visualize the traffic light status at an intersection. Grafana is a versatile open-source analytics and interactive visualization web application that enables the creation of informative charts, graphs, and alerts when connected to compatible data sources. To construct this traffic light visualization, we utilized the "Traffic Lights" plugin available in Grafana. This plugin facilitates the

design of a simple layout representing a four-lane intersection. The associated SQL query connects to a PostgreSQL database, filters the relevant lane light statuses, and then displays the corresponding light colors on the traffic light plugin. This functionality proves valuable for emulating a traffic management center's role, allowing for the real-time monitoring of individual intersection light statuses.

The Plugin with the associated SQL Query and the Grafana Dashboard is shown in Figure 48 and Figure 49. The SQL files accociated with the dashboard can be found in *Grafana* folder in the Githib repo.

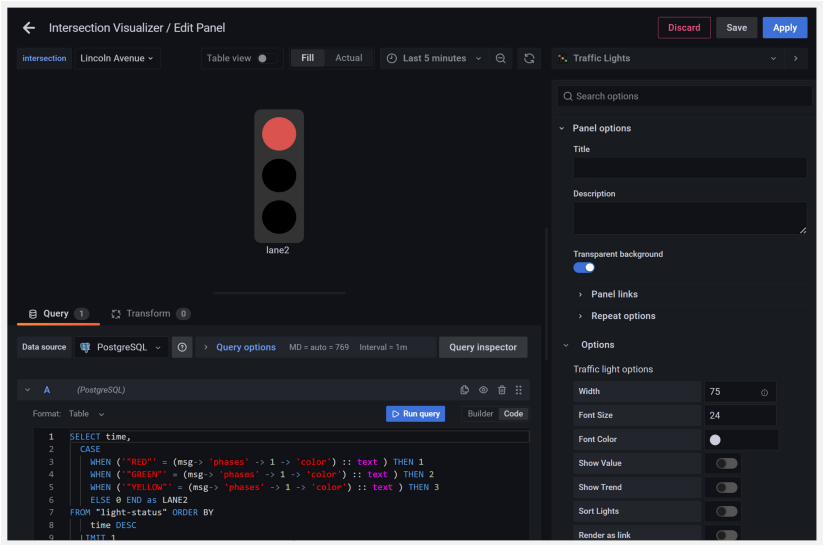

Figure 48: Grafana Traffic Light Plugin

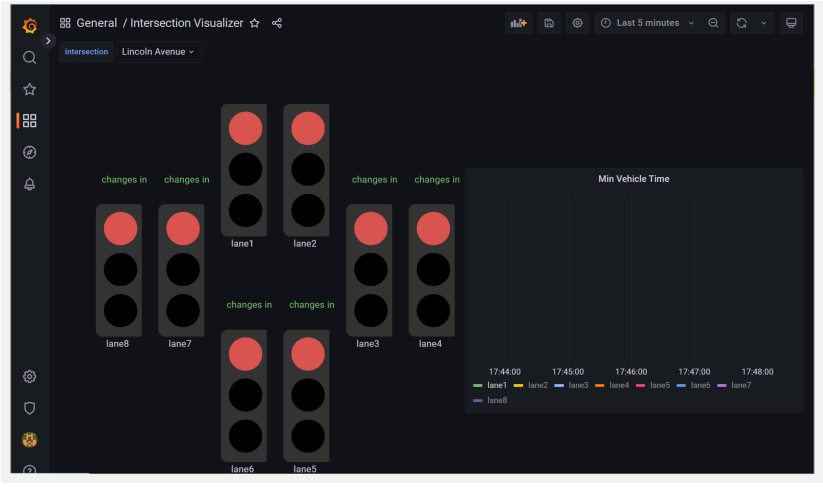

Figure 49: Grafana Dashboard

4.9 Diode Component Enclosures

Two enclosures were meticulously designed to house the two STM32 boards and the cell modem, each undergoing iterative improvements for optimal functionality and aesthetics.

The first version, labeled as version 1.0, utilized a waterproof box from Polycase, specifically the WQ-48 Hinged Grey Waterproof Box with a transparent lid. The dimensions of this box were 8.24 x 6.29 x 3.93 inches. The choice of this box was based on factors such as its size, transparent lid, and compatibility with the diode layout. More details about this box can be found on the Polycase website https://www.polycase.com/wq-48.

The transparent lid of the box was particularly essential as it allowed the LED status indicators on the world side STM32 board to be visible from the outside. This way, observers could monitor the transmission status without needing to open the enclosure.

A custom mounting plate was crafted using acrylic material with the assistance of the Purdue ECE mechanical shop. This mounting plate was affixed to the bottom of the box, and it featured strategically positioned holes, screws, and nuts to securely hold both STM32 boards and the cell modem in place.

The assembly process began by mounting the TSC side nucleo board onto the plate. Subsequently, the world side STM32 board was stacked on top of the TSC side board, with insulation material in between to prevent any electrical interference or contact issues. The Cell Modem was fixed to the plate in the space beside the stacked boards. This arrangement ensured that the LEDs on the world side board and the cell modem were directly visible through the transparent lid, offering a convenient means of monitoring the data transmission status.

To facilitate the connection between the TSC side nucleo board and the external network (Econolite or PC), a 1-inch hole was drilled in the enclosure in front of the ethernet port of the TSC side nucleo board. This aperture allowed for the easy insertion of an Ethernet cable for network connectivity.

A separate 5V USB hub was incorporated inside the enclosure to individually power both nucleo boards and the cell modem. This hub was also contained within the enclosure to maintain a clean and organized setup. The USB power cable from the hub was routed through another hole in the enclosure, effectively creating a plug-and-play system.

This enclosure design and assembly process ensured the robustness of the system while allowing for easy monitoring, maintenance, and power management. The arrangement of components, transparent lid, and well-thought-out cable management contributed to the overall functionality and visual appeal of the enclosure.

The version 1.0 enclosure is shown in Figure 50.

An alternative version, denoted as version 2.0, featured a more compact enclosure design that eliminated the need for wiring between components. This version embraced a miniature approach, with both STM32 boards and the cell modem mounted onto a single PCB plate. The KiCAD files required for fabricating the PCB can be accessed within the GitHub repository.

For this version, a 3D-printed enclosure was designed by David Hong from Virginia Tech. The CAD files for this enclosure design are also available in the GitHub repository under the folder **Diode_Enclosure_Version2.0**. Notably, the enclosure

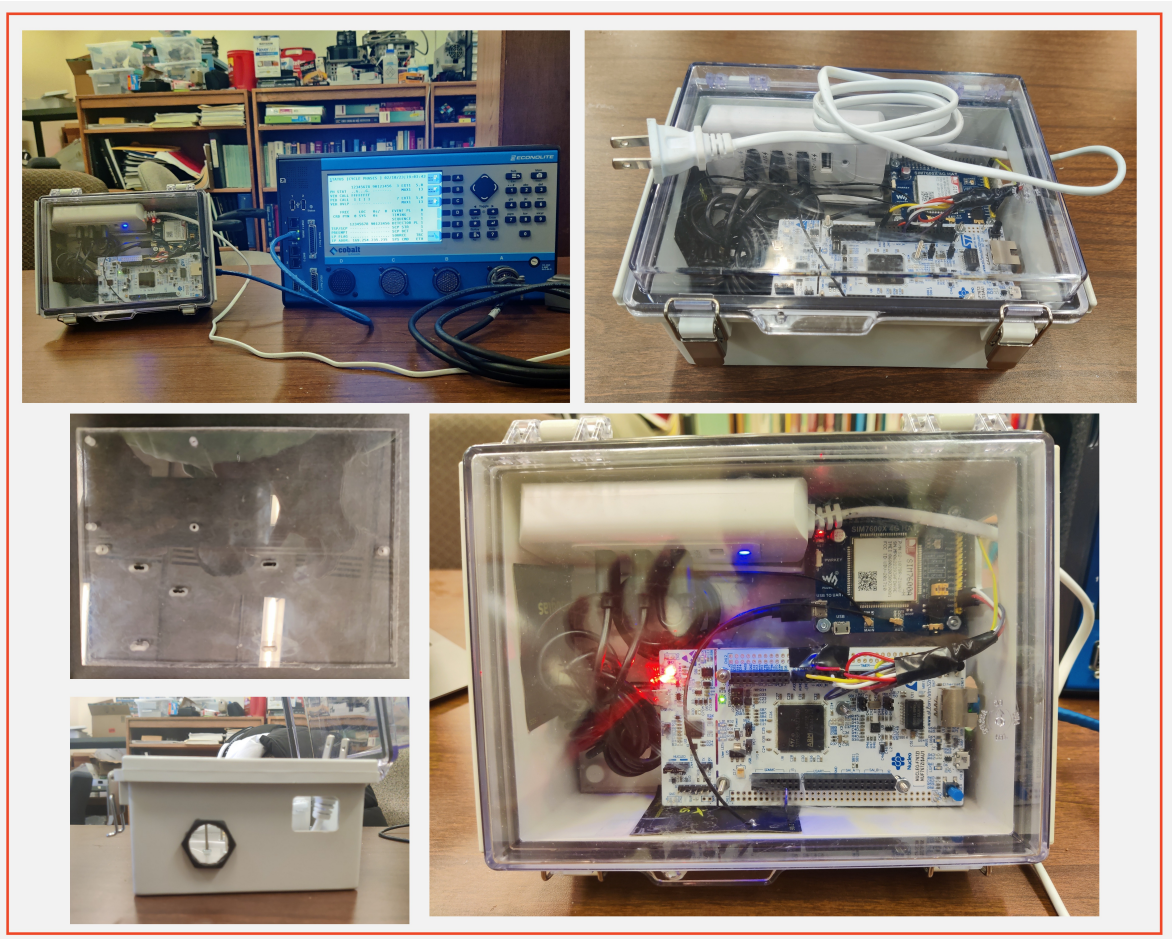

Figure 50: Diode Enclosure Version 1.0

was tailored to accommodate the no-wire system, emphasizing a clean and organized setup.

Unlike the previous version, the only cable involved in this configuration is the power cable connecting to the Green Terminal Blocks, responsible for supplying power to the PCB. This streamlined approach minimized clutter and enhanced the aesthetic appeal of the system.

The dimensions of version 2.0 were considerably smaller than version 1.0, measuring 15cm x 14.5cm x 7cm. Additionally, the enclosure size could be further reduced by removing the programmer used to program the STM32 boards. In such cases, the boards could be programmed externally and then connected to the PCB using wires. This is a feature of STM32 Nucleo boards.

A notable aspect of this version is the jumper pins for the power supply arrangement. The JP3 jumper on both STM32 boards needed to be in the **E5V** position. It's important to emphasize that when the jumper is set to this position, the code cannot be uploaded to the boards using the USB interface. Code uploading requires switching the jumper to the **U5V** position. After uploading the code, the jumper should be returned to the **E5V** position for normal operation. This is a common mistake made by programmers during debugging. Further the UART selection jumper on the cell modem needs to be in position A, which is meant to *"access raspberry pi via USB to UART"*.

Both versions of the enclosure were designed to fit within the confined space of the traffic cabinets in Owosso. These designs aimed to strike a balance between functionality, size, and ease of installation, ensuring a seamless integration into the existing infrastructure.

The version 2.0 enclosure is shown in Figure 51.

4.10 Field Testing and Deployment Experiences: Diode System in Michigan Traffic Management

Field Testing and Deployment Experiences: Diode System in Michigan Traffic Management

The first visit took place on March 8, 2023, when the initial version of the system, version 1.0, was delivered by Purdue ECE to MDOT in Lansing. This marked the commencement of the bench testing phase. The diode system, equipped with version 1.0, performed successfully for an entire week. However, a sudden interruption occurred as the system unexpectedly ceased transmitting data to the NATS server. After thorough investigation, the root cause was identified: the modem had lost cellular signal. The situation was rectified by performing a system reset, which reinstated the broadcasting functionality. Following this incident, the diode continued to operate reliably, and its performance remained consistent even when the enclosure door was closed. It was revealed that certain networks, notably Verizon, were experiencing connectivity issues. The testing phase utilized a SIM card provided by Purdue OATS.

The second visit to Michigan transpired on August 7, 2023. During this visit, both versions of the diode system were set up in the Lansing signal shop and subsequently

Figure 51: Diode Enclosure Version 2.0

deployed at two intersections in Owosso. The intersections agreed to by MDOT were:

(M52 x King St.)
Roughly: 43°00'17.8"N 84°10'36.6"W
43.004956, -84.176830
&
(Washington x M21(Main))
Roughly: 42°59'51.0"N 84°10'14.4"W
42.997511, -84.170679

The installations at these intersections were aimed to assess the practical performance of the diode system under real-world conditions. The deployment was instrumental in evaluating the effectiveness of both version 1.0 and version 2.0 enclosures, as well as the overall functionality of the system.

These visits underscored the significance of real-world testing and fine-tuning the system in response to unexpected challenges. It was further discovered that the mapping from lanes to signal group needed a revisiting. The insights gained from these experiences contributed to the refinement and optimization of the diode system, ultimately enhancing its reliability and suitability for deployment in traffic management scenarios. A still from the second visit is shown in picture 52.

Figure 52: A still from second visit to Michigan

4.11 3rd annual student-run Next-generation Transport Systems Conference (NGTS)

The data diode system was prominently featured at the 3rd annual student-run Next-generation Transport Systems Conference (NGTS), held from May 16 to 18. Purdue ECE took the opportunity to conduct an in-person presentation, effectively showcasing the project's accomplishments to a diverse audience of conference attendees. The presentation provided an insightful overview of the diode's functionality, integration, and potential applications in modern transport systems.

The presentation was met with appreciation and engagement from the conference attendees, indicating a keen interest in the innovative solution presented. This positive response underscores the project's significance and its relevance to the field of transportation technology.

To further enhance the dissemination of the project's findings, the presentation file has been made available in the NTGS_Conference folder within the project's GitHub repository.

This participation in a renowned conference adds another dimension to the project's success, enabling the broader transportation community to gain insights into the system's capabilities and potential impact on traffic management practices. A moment from the conference is captured in the accompanying picture 53

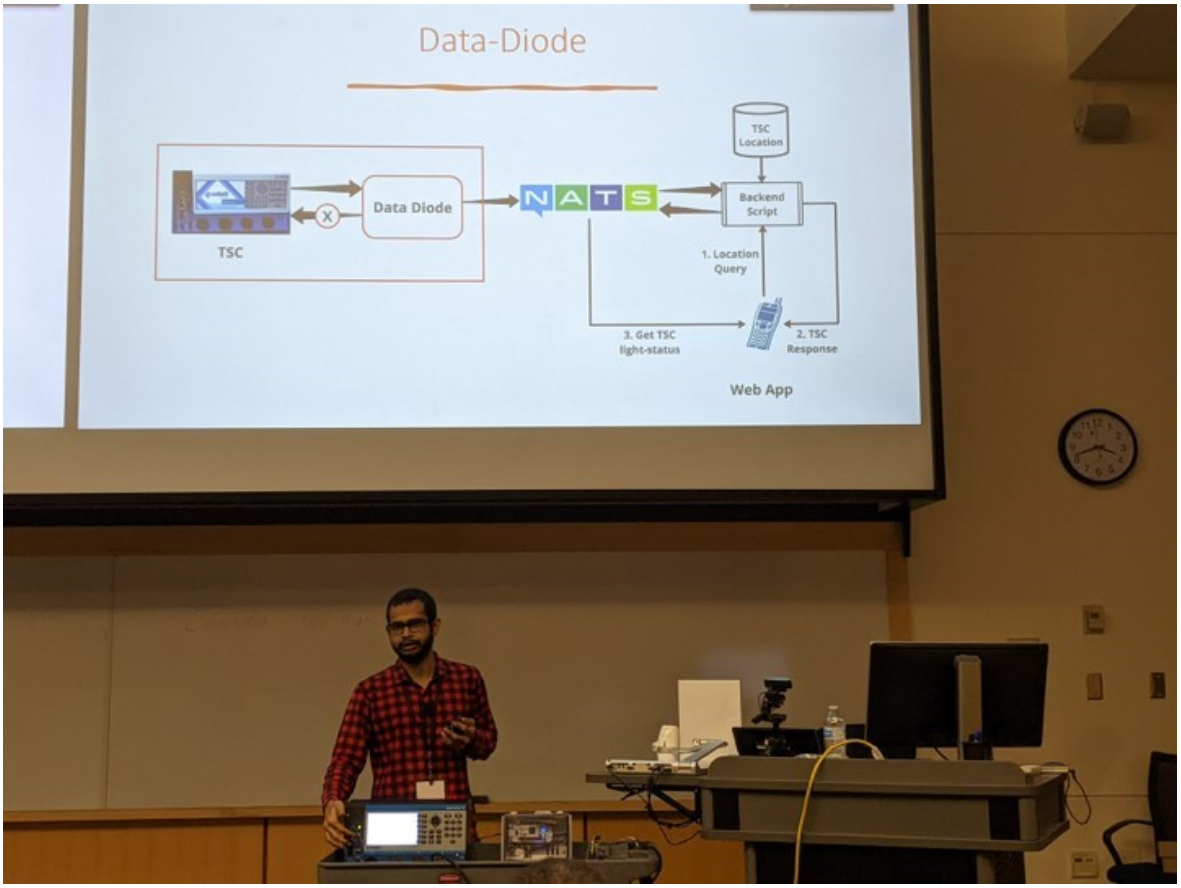

Figure 53: A still from demonstration at the NTGS Conference

5. Results

The deployment of the data diode system in the context of Michigan's traffic management yielded valuable insights and demonstrated the system's functionality in various scenarios. The diode system, consisting of STM32 Nucleo boards, cellular modems, and specialized software components, was rigorously tested and showcased its capabilities through field visits and bench testing.

During the initial phase of the project, the system was extensively tested to ensure its accurate functioning. A crucial aspect was the establishment of seamless communication between the diode, the NATS server, and the user interface application. The utilization of the NATS messaging system proved to be advantageous due to its subject-based addressing mechanism, which facilitated data flow and simplified the interaction between the various components.

The system exhibited robust performance in practical operational scenarios, transmitting Signal Phase and Timing (SPaT) data from traffic controllers to the NATS server. The use of a data diode allowed the unidirectional flow of information, ensuring that critical infrastructure remained protected from potential external threats.

One of the key achievements of the project was the successful deployment of two diode systems in the city of Owosso, Michigan. These diodes were strategically positioned within traffic cabinets at two different intersections. The deployment process involved careful configuration, placement, and connectivity setup to enable smooth data transmission. Throughout the deployment phase, the diodes effectively transmitted SPaT data to the NATS server, contributing to real-time traffic analysis and decision-making.

Latency measurement emerged as a critical aspect of the system's performance evaluation. The latency measurement component was integrated into the diode system, allowing the calculation of uplink latency and end-to-end latency. The measurement revealed the average latency and its variance, providing crucial information for accurate prediction of future light states. This data allowed for a more informed understanding of traffic patterns and trends.

The utilization of Svelte, a modern web application framework, facilitated the development of an intuitive user interface. The application enabled users to interact with the diode system, receive real-time traffic updates, and access SPaT data for different intersections. The app's accuracy was subject to the precision of GPS coordinates, and while the system provided valuable insights, it is important to account for the inherent limitations of GPS technology.

Furthermore, the integration of MAP data units and postgress databases enhanced the system's capabilities. By mapping unique IDs to specific intersections, the system efficiently directed SPaT data to relevant subjects. This allowed users to subscribe to intersections of interest and receive accurate, location-specific traffic information. NATS framework minimised the downlink latency of the system.

Through field visits and extensive bench testing, the data diode system demonstrated its reliability, adaptability, and potential to enhance traffic management prac-

tices. The successful deployment in Michigan showcased the system's seamless integration into existing traffic infrastructure and its capability to provide real-time, accurate traffic data.

Through field visits and extensive bench testing, the data diode system demonstrated its reliability, adaptability, and potential to enhance traffic management practices. The successful deployment in Michigan showcased the system's seamless integration into existing traffic infrastructure and its capability to provide real-time, accurate traffic data.

6. Discussion

The overall Components of the system include 2 STM32 Nucleo-144 Boards, A cell modem, 2 Antennas, an Ethernet cable, 2 USB cables and 7 jumper wires

The estimated cost of the system (ignoring the cost of the cables) is :

Component	Unit x Price	Total Cost in $
STM32	2x23	46
Cell Modem	1x71	71
Antennas	2x11	22
Polycase box	1x40	40
Mounting Plate	1x20	20
USB hub	1x10	10
PCB	1x10	10
3D Printing Box V2.0	1x20	20
Total (Verison 1.0)	-	209
Total (Version 2.0)	-	169

Version 2.0 of the system presents a cost-effective and streamlined solution compared to Version 1.0. In terms of initial costs, Version 2.0 is already more affordable than its predecessor. However, the cost advantages become even more significant when the system is scaled up and deployed on a larger scale.

The reduced cost of Version 2.0 can be attributed to its efficient design and the elimination of certain components present in Version 1.0. The integration of the two STM32 boards and the cell modem onto a single PCB plate reduces the need for additional wiring and connectors, resulting in lower manufacturing and assembly costs. Additionally, the use of a 3D-printed enclosure for Version 2.0 simplifies the manufacturing process and reduces material costs.

One of the notable advantages of Version 2.0 is its compact design and minimalistic setup. The absence of wires and the USB hub in Version 2.0 contributes to a hassle-free deployment process. Without the need for additional components like USB hubs and separate power cables, the installation becomes more straightforward and less time-consuming. This feature becomes particularly advantageous when deploying the system across multiple intersections or locations, where simplicity and efficiency are crucial.

Furthermore, the smaller footprint of Version 2.0's enclosure makes it easier to accommodate within traffic cabinets or other designated spaces. The reduced space requirement aligns well with the spatial constraints often encountered in such environments.

In conclusion, Version 2.0 offers a cost-effective, space-efficient, and simplified solution for data transmission in comparison to Version 1.0. As the system is scaled up for widespread deployment, the economies of scale and the efficiency of the design are expected to lead to even more significant cost reductions and smoother implementation processes.

7. Conclusion

The development and implementation of the data diode system represent a significant step forward in enhancing traffic management and control systems. Through the seamless integration of hardware and software components, we have successfully created a robust and reliable solution that addresses the challenges of secure data transmission across traffic signal controllers. The project's multifaceted approach, encompassing hardware design, software development, communication protocols, and real-world deployment, underscores its comprehensive nature.

The achieved outcomes include the establishment of a bidirectional communication channel between traffic signal controllers, the integration of NATS messaging protocol for efficient and reliable data exchange, and the utilization of GPS data to predict and display real-time traffic signal status. The system's ability to provide accurate and up-to-date traffic information to users has the potential to significantly improve road safety, congestion management, and overall transportation efficiency.

Given the low cost of the system, the results of the realised simplex communication based transmission and reception are encouraging and the delivered NTCIP-Based SPaT data to the end user through NATS provides large scope for enhanced user experience, data processing and analytics.

While the project has demonstrated successful implementation and functionality, there are avenues for future work and enhancement. First, refining the accuracy of GPS data and ensuring a consistent signal reception will further improve the reliability of traffic signal predictions. Additionally, exploring the integration of latency in displaying the light staus with minimal lag could enhance the system's predictive capabilities and adaptability to changing traffic scenarios.

Furthermore, expanding the system's deployment to additional intersections and urban environments will help assess its scalability and robustness in various traffic conditions. Evaluating the system's performance under high-density traffic scenarios, adverse weather conditions, and potential network disruptions will provide valuable insights for system optimization.

Proper handling of data reading from the TSC when the Department of Transportation (DOT) is utilizing the SERVER IP has to be studied further. Investigating

the intricacies of this process and implementing a robust solution will enhance the system's compatibility and acceptance by the DOT.

A comparative analysis of different communication protocols, like SPI (Serial Peripheral Interface), UART, and I2C (Inter-Integrated Circuit), can be conducted to assess the extent of data loss within each of these systems. This analysis aims to understand the efficiency, reliability, and performance of these communication methods when transmitting data.

During the data transmission process from the diode, it's important to consider the optimization of data payload. Specifically, when transmitting data from an intersection with only a few active signal groups, there's no need to include data related to signal groups that are not relevant. This strategic approach can significantly contribute to data efficiency and conservation.

By excluding data pertaining to inactive or irrelevant signal groups, the overall data payload can be reduced. This reduction not only saves data but also enhances the efficiency of communication between the diode and the receiving system, such as the NATS server or other connected devices. This approach is particularly beneficial in scenarios where bandwidth or transmission resources are limited, as it minimizes the amount of data that needs to be processed and transmitted.

Implementing a selective data inclusion strategy requires intelligent data filtering and management on the diode side. This can involve assessing the status of each signal group and determining whether it's currently active or inactive. Active signal groups, which are the ones currently affecting traffic patterns, will have their data included in the transmission. Inactive signal groups, on the other hand, can be skipped to reduce unnecessary data overhead.

Incorporating user feedback and fine-tuning the user interface of the accompanying mobile app can enhance its usability and accessibility, ensuring that end users can easily interact with and benefit from the real-time traffic information provided.

In conclusion, the data diode system marks a significant stride towards smarter and more efficient traffic management. By addressing the challenges of secure data transmission, real-time monitoring, and predictive analysis, the system demonstrates its potential to revolutionize traffic control systems. The project's successful execution opens the door to further innovation and collaboration in the domain of transportation technology, ultimately contributing to safer, more reliable, and more effective road networks.

A. Appendices

A.1 NTCIP Object Definitions for Actuated Traffic Signal Controller (ASC) Units - version 02

https://tinyurl.com/54hx86pe

A.2 STM32 Libraries

1. Ethernet Library : https://github.com/stm32duino/STM32Ethernet
2. Arduino Boarld URLs : https://tinyurl.com/bddus8sz

A.3 STM32 F767ZI Nucleo-144 Manuals

1. Website : https://www.st.com/en/evaluation-tools/nucleo-f767zi.html
2. Pinout : https://os.mbed.com/platforms/ST-Nucleo-F767ZI/
3. User Manual : https://tinyurl.com/ycxs4v55
4. Reference Manual : https://tinyurl.com/2t2p5y7s

A.4 Cell Modem and Antenna

1. Wiki Page : https://www.waveshare.com/wiki/SIM7600E-H_4G_HAT
2. AT Command Manual : https://tinyurl.com/4ause5ef
3. Antenna Datasheet : https://tinyurl.com/3t57zpj4

A.5 Ethernet capture setup

https://wiki.wireshark.org/CaptureSetup/Ethernet

B. References

[1] VA Arroyo, SE Bennett, DH Butler, M Dougherty, A Stewart Fotheringham, JS Halikowski, A Dot, PW Michael Hancock, S Hanson, S Heminger, et al. Nchrp report 812–signal timing manual, 2015.