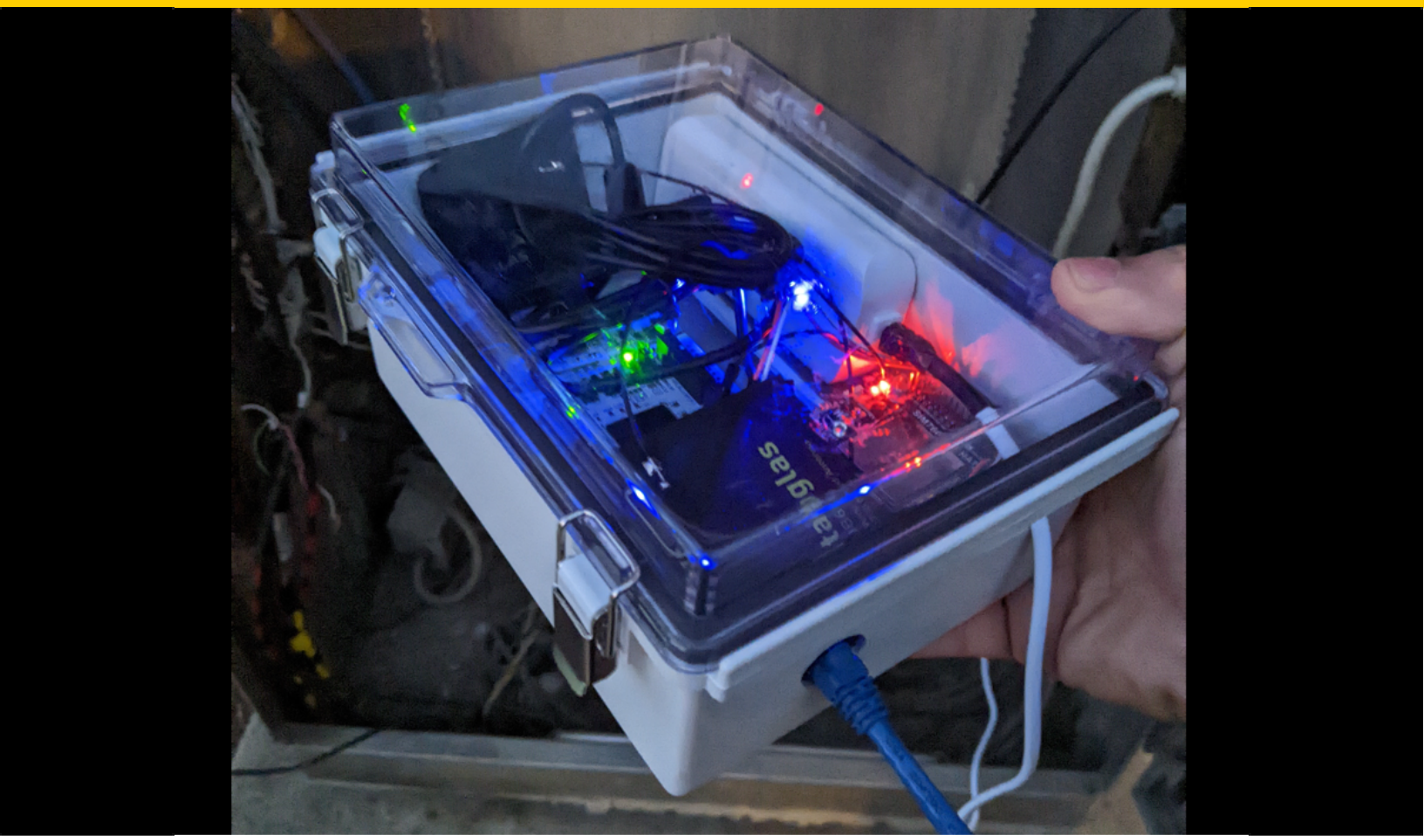
CENTER FOR CONNECTED AND AUTOMATED TRANSPORTATION

Final Report #71
November 2023



Economical Acquisition of Intersection Data to Facilitate CAV Operations

Manish Gowda
Walt Fehr
Andrew Balmos
Richard Ajagu
James Krogmeier
Montasir Abbas
Samuel Labi

PURDUE UNIVERSITY®

CENTER FOR CONNECTED AND AUTOMATED TRANSPORTATION

Report No. 71
Project Start Date: January 2022
Project End Date: September 2023

November 2023

Economical Acquisition of Intersection Data to Facilitate CAV Operations

Manish Gowda
Graduate Researcher

Walt Fehr
Engineer

Andrew Balmos
Graduate Researcher

Richard Ajagu
Graduate Researcher

James Krogmeier
Professor

Montasir Abbas
Professor

Samuel Labi
Professor

Purdue University
Volpe Center
Virginia Polytechnic Institute & State University

ACKNOWLEDGEMENTS AND DISCLAIMER

Funding for this research was provided by the Center for Connected and Automated Transportation under Grant No. 69A3551747105 of the U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology (OST-R), University Transportation Centers Program. The authors acknowledge the support provided by the following team members: the City of Owosso Mayor Christopher Eveleth, Messrs. Mark Goodfriend, David Acton (President of The Transformation Network, Inc.), Tim Johnson (T-Mobile), Sandy Klausner (Cubicon Inc.), and Michigan DOT Engineers (Messrs. Nathan Bouvy, Phillip Travis, Hilary Owen, Scott Holzhei, Ryan Schian, Ross Venable, and James Kwapiszewski), and David Hong of VirginiaTech, and Dr. Darcy Bullock (Purdue). The organizations, institutions and professors that provided valuable in-kind cost share and other support for this project are also acknowledged. Also, general support was provided by the following initiatives at Purdue University's College of Engineering: Institute for Control, Optimization, & Networks (ICON) and the Autonomous & Connected Systems (ACS). The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated under the sponsorship of the Department of Transportation, University Transportation Centers Program, in the interest of information exchange. The U.S. Government assumes no liability for the contents or use thereof.

Suggested APA Format Citation:
Gowda, M.K., Fehr, W., Balmos, A., Ajagu, R., Krogmeier, J., Abbas, M., Labi, S. (2023). Economical Acquisition of Intersection Data to Facilitate CAV Operations, CCAT Report #71, Center for Connected and Automated Transportation, Purdue University, West Lafayette, IN.

Cover image credit: Richard Ajagu.

Contacts

For more information:

Samuel Labi, Ph.D.
550 Stadium Mall Drive
HAMP G167B
Phone: (765) 494-5926
Email: labi@purdue.edu

CCAT
University of Michigan Transportation Research Institute
2901 Baxter Road
Ann Arbor, MI 48152

uumtri-ccat@umich.edu
(734) 763-2498
www.ccat.umtri.umich.edu

Technical Report Documentation Page

1. Report No. 71	**2. Government Accession No.**	**3. Recipient's Catalog No.**
4. Title and Subtitle Economical acquisition of intersection data to facilitate CAV operations		**5. Report Date:** November 2023 **6. Performing Organization Code:** N/A
7. Author(s) Manish Gowda, Walt Fehr, Andrew Balmos, Richard Ajagu, James Krogmeier, Montasir Abbas, Samuel Labi		**8. Performing Organization Report No.** N/A
9. Performing Organization Name and Address Center for Connected and Automated Transportation Purdue University, 550 Stadium Mall Drive, W. Lafayette, IN 47907; and Univ. of Michigan Ann Arbor, 2901 Baxter Rd, Ann Arbor, MI 48109		**10. Work Unit No.** **11. Contract or Grant No.** Contract No. 69A3551747105
12. Sponsoring Agency Name and Address U.S. Department of Transportation, Office of the Assistant Secretary for Research and Technology, 1200 N.J. Ave., SE, Washington, DC 20590		**13. Type of Report and Period Covered:** Final Report, 1/1/2022 - 06/30/2023 **14. Sponsoring Agency Code:** OST-R

15. Supplementary Notes

Conducted under the U.S. DOT Office of the Assistant Secretary for Research and Technology's (OST-R) University Transportation Centers (UTC) program.

16. Abstract

Cost-effective ways to obtain and distribute traffic data, and thereby, to facilitate operations decisions. This includes a need to make Signal Phase and Timing (SPaT) and MAP data more useful to traffic data end-users. Existing data collection and delivery methods are time consuming and costly. Therefore, to facilitate traffic operations in the current era of human-driven vehicles (HDVs) and the prospective era of connected autonomous vehicles (CAVs), this research developed a device to acquire, process, and disseminate SPaT data from Traffic Signal Controller (TSC) cabinets in a manner that ensures system integrity and yet fosters accessibility of timing information for traffic data end-users. The device, referred to as a "Data Diode," consists of two interconnected microcontrollers operating through simplex communication (one-directional data flow). The device connects the TSC cabinet and end-user devices (such as a user-held device (UHD)). The process begins with one microcontroller connecting to the TSC's Ethernet port. Then, using the Simple Network Management Protocol (SNMP) interface, the device securely acquires the SPaT data which then undergoes Cyclic Redundancy Check (CRC) encoding to ensure integrity protection and subsequently transmitted through a unidirectional UART interface to the second microcontroller. No SNMP commands can be passed back to the signal controller from the Internet. The interface cannot be used as an attack surface. 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. A battery of such TSC-Diode systems collect such data from various intersections, and the collected data are routed through their respective channels within the NATS system. A backend script acquires GPS coordinates from the UHD, 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 UHD. The app dynamically presents real-time information about the traffic signal status and precise timing of signal phase transitions for the lane of interest. Overall, the integration of this technology into human-driven or connected autonomous vehicle driver assistance systems can help smoothen arterial traffic flow and transform urban mobility.

17. Key Words Data diode, Traffic signal controller, User datagram protocol, Cyclic redundancy check, Signal phase & timing.	**18. Distribution Statement** No restrictions.

19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages** 92	**22. Price**

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ATSC	Actuated Traffic Signal Controller
CAV	Connected and Autonomous Vehicle
CLI	Command Line Interface
CRC	Cyclic Redundancy Check
HDV	Human Driven Vehicle
IDE	Integrated Development Environment
ITS	Intelligent Transportation Systems
JSON	JavaScript Object Notation
MAP	Intersection Map
MDOT	Michigan Department of Transportation
MIB	Management Information Base
NATS	Neural Autonomic Transport System
NCTIP	National Transportation Communications for ITS Protocol
RAT	Radio Access Technology
RPC	Remote Procedure Call
SDN	Software-Defined Networking
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SPaT	Signal Phase and Timing
TCP	Transmission Control Protocol
TSC	Traffic Signal Controller
TSCBM	Traffic Signal Controller Broadcast Messages
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UHD	User Handheld Device

CHAPTER 1 INTRODUCTION

1.1 Study Background and Problem Statement

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 [1]. 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 was evident a few years ago in Gwinnett County, Georgia, where equipping a mere 20 intersections with Dedicated Short-Range Communication (DSRC) equipment incurred an exorbitant cost. Such high costs underscore the imperative for more economical approaches to obtain and distribute essential data for intersections.

Further, 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 cost to equip urban intersections with computing devices and short-range wireless communication can be prohibitive. In the example given above for Gwinnett County, Georgia, it cost $309,000 to equip 20 intersections with dedicated short-range communication (DSRC) equipment. The Master Plan for Connected Vehicle Technology in Gwinnett contains details of cost data regarding DSRC implementation. The Gwinnett projects were part of a Smart Corridor GDOT-collaboration project involving deployments of onboard units, roadside units, and test software to support implementation of the master plan. The objective was to deploy DSRC at ninety-two (92) state route intersections including thirty-six (36) intersections on SR 141 and SR 140 corridors. The deployments in the Georgia projects included railroad intersection blocked alerts, signal phase & timing information, transit signal priority, emergency vehicle preemption, construction and maintenance vehicle alerts, and pedestrian presence alerts. Table 1.1 presents approximate deployment costs of the system elements. The cost estimate for RSU includes the cost of the intersection mapping, RSU purchases and field installation. The unit costs shown are averaged over 20 intersections.

Table 1.1 Costs of Gwinnett County's Deployment

(a) Connected Vehicle Hardware Costs (Includes purchase and installation)

Device	Device with installation estimate
DSRC OBU System	$1,000 – $3,000/unit system
DSRC RSU Systems	$4,000 – $6,000/intersection
C-V2X Device	$6,000 – $7,000/intersection
Edge Device	$250 – $450/intersection
HMI	$1,000 – $3,000/unit device

(b) Sample Project Field Installations

Field Installation	# of units + Install + Programming	Unit Cost	Total Cost
Cybersecurity (SCMS)	20	$100	$2,000
Edge processing	20	$350	$7,000
Roadside Unit	20	$5,000	$100,000
Service (units = years)	3	$10,000	$30,000
MAP Message Development	20	$1,500	$30,000
Software (units=intersections)	20	$7,000	$140,000
TOTAL			$309,000

(c) Sample Project Vehicle Installations

Field Installation	# of units + Install + Programming	Unit Cost
OBU	20	$3,000
Cybersecurity (SCMS)	20	$100
HMI	20	$5,000

Cost-effective ways are needed to obtain and distribute traffic data, and thereby, to facilitate operations decisions. This includes a need to make Signal Phase and Timing (SPaT) and MAP data more useful to traffic data end-users. Empirical evidence suggests that the existing data collection and delivery methods are time consuming and costly.

Therefore, to facilitate traffic operations in the current era of human-driven vehicles (HDVs) and the prospective era of connected autonomous vehicles (CAVs), a system is needed to acquire, process, and disseminate SPaT data from Traffic Signal Controller (TSC) cabinets in a manner that ensures system integrity and yet fosters accessibility of timing information for traffic data end-users.

1.2 Study Objectives

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 pursued through the development and deployment of a cutting-edge, low-cost data diode solution.

1.3 Study Framework

The installation in Georgia used the traditional way of gathering and distributing SPaT and MAP data using computing and short-range wireless equipment at each intersection. Figure 1.1 presents the physical view diagram for an alternative way of partitioning the system to distribute SPaT and MAP data using a simpler way of connecting to the traffic signal controller, and then using edge devices to create digests of that data for more efficient delivery to the mobile objects of the system.

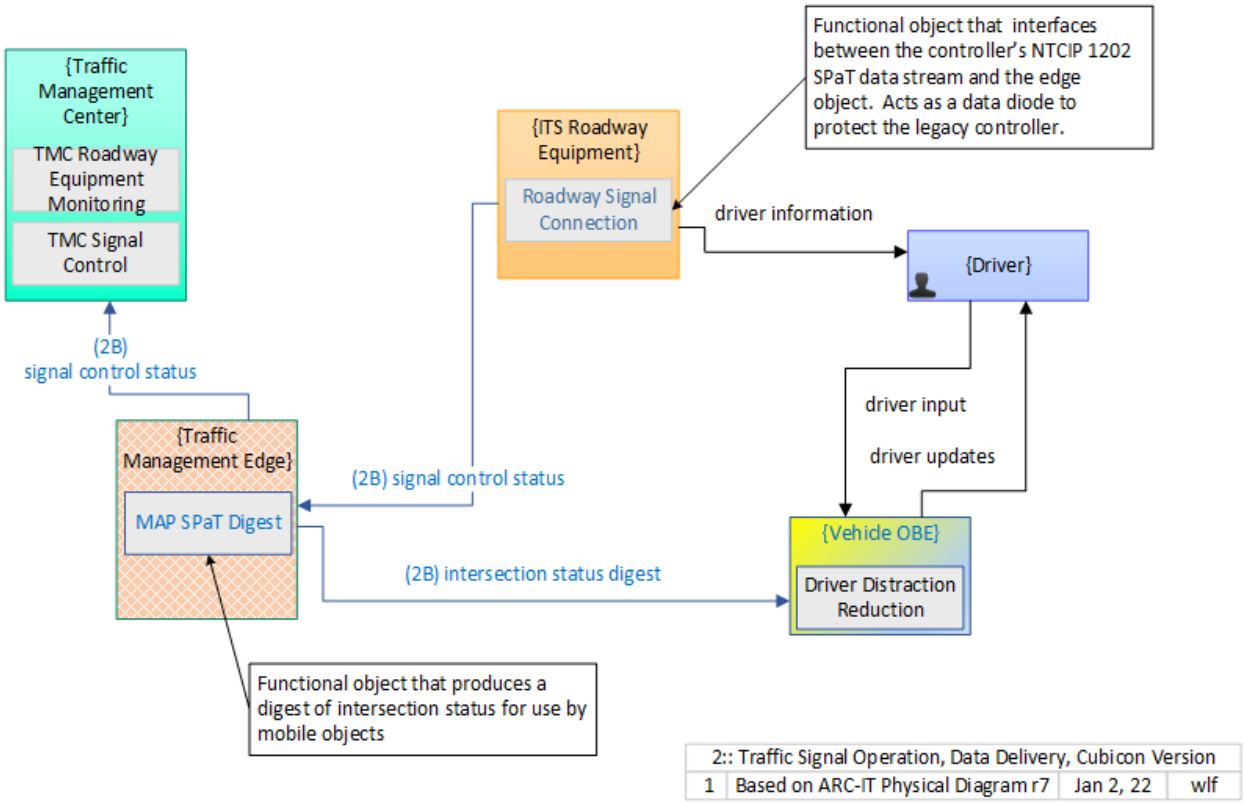


Figure 1.1: Physical view diagram for partitioning the system to distribute SPaT and MAP data, as conducted in the current study

In the field physical object {ITS Roadway Equipment} on this physical view diagram served as an opportunity to create a high-integrity data diode. The unidirectional diode was developed to be capable of delivering the state data stream from a legacy traffic signal controller to the edge object using Internet protocols over the wide-area network where the digest will be maintained. This was done while protecting the legacy controller from any harm that may come from the Internet. This opportunity will be used to test the idea of creating a low-cost, high-integrity device.

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 safe-guarded 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.

1.4 The Study Team

The study team consisted of representative from the three key stakeholders: government (Michigan DOT, City of Owosso), industry (T-Mobile, The Transformation Network, Inc., CubeFog, Mark Goodfriend Inc.), and academic (Purdue University, Virginia Tech). These participants worked together to design the study product (that is, the data diode device) and to plan the pilot deployments of the device.

1.5 Organization of this Report

The report first presents the study background and motivation, the study objectives and scope, and the corporate members of the study team. This is followed by a description of the overall study approach (Chapter 2), the systems architecture (Chapter 3) and the components of the code developed to accomplish the study objective (Chapter 4). Chapter 5 describes the device development and the details and results of the field deployment and testing processes, and Chapter 6 offers some concluding remarks including the practical usefulness of the device developed in this project, and avenues for prospective deployment and further device development. Chapter 7 summarizes the levels of established USDOT performance indicators achieved in this study. Chapter 8 presents the study outputs, outcomes, and impacts.

CHAPTER 2 GENERAL STUDY APPROACH

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 current and future movements orchestrated by one or more signal controllers. Beyond SPaT, these messages also furnish insights into lane or intersection regulations that fluctuate with the time of day, such as designating a lane to one direction in peak traffic hours. Appendix A.1 provides a link to the standard NTCIP Object Definitions for Actuated Traffic Signal Controller (ATSC) 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 [2] , each signalized intersection possesses 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 presented in Figure 2.1. To further illuminate the interplay between these signalized elements and the users of the system, Figure 2.2 illustrates the essential interactions.

The detectors, encompassing vehicle, pedestrian, or bicycle sensors, play a pivotal role by transmitting messages to the controller as and when needed by the end user (for example, when the driver approaches 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 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 near intersections, housing the vital electronic controls needed for traffic signal operations. These cabinets serve as repositories of crucial data, pivotal for informed decision-making and thorough analysis. However, the acquisition process must not take place 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 are 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 2.3 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.

The data diode concept represents 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 current project seeks to establish a robust solution that maintains affordability while safeguarding data velocity and veracity.

In response to this challenge, the current project seeks to use a twin-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.

SPaT data are wirelessly transmitted by the cell modem interfacing the second microcontroller to a NATS Server, where it is 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.

Figure 2.1: Typical Signalized Intersection Components [6]

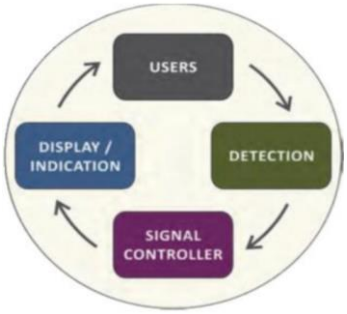


Figure 2.2: Generic Signalized Intersection Interaction (anon)

Figure 2.3: Typical TSC Cabinet (anon)

Furthermore, the project represents a practical application that can enhance the convenience or experience of the end user. 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. In subsequent sections of this report, we provide comprehensive details of the project, spanning its constituent hardware and software elements to the workflow setup and operational methodologies. Through this multifaceted endeavor, the project seeks to redefine the landscape of intersection data acquisition and security in the context of urban traffic operations. This, hopefully, will set a precedent for efficient, secure, and user-centric urban transportation systems.

CHAPTER 3 SYSTEM ARCHITECTURE

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

3.1 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 Universal Asynchronous Receiver-Transmitter (UART) connection.

3.2 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".

3.3 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 re publishing 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.

3.4 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.

3.5 Integration

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 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 outlined the diverse components in conjunction with their interconnections that constitute the Data Diode system. Figure 3.1 presents this comprehensive architecture (synergistic integration of the components). Furthermore, Figure 3.2 presents the practical manifestation of the system and indicates a real-life demonstration. Notably, the initial rendition of the data diode, Version 1.0, is showcased in Figure 3.2.

Subsequently in this report (Chapter 5), we describe in detail, these interconnections, interdependency features, as well as the operational intricacies of each individual component, a comprehensive exposition. Therefore, Chapter 5 provides an understanding of the seamless orchestration that underpins the Data Diode system's functionality.

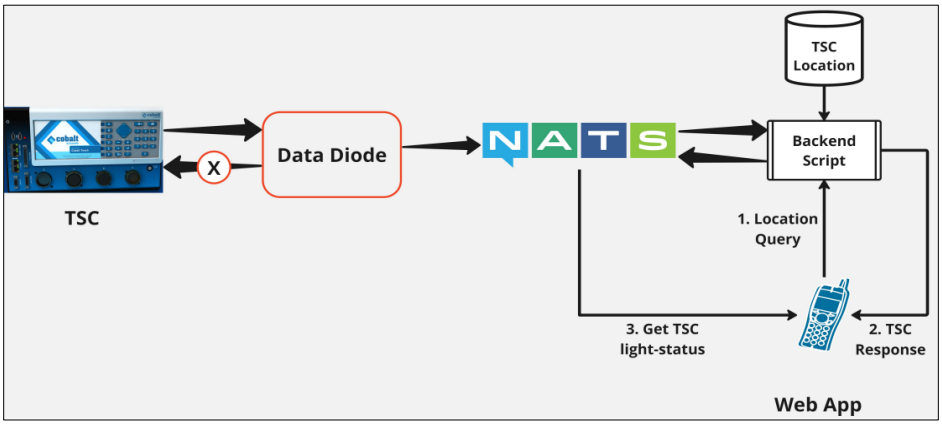


Figure 3.1: System Workflow

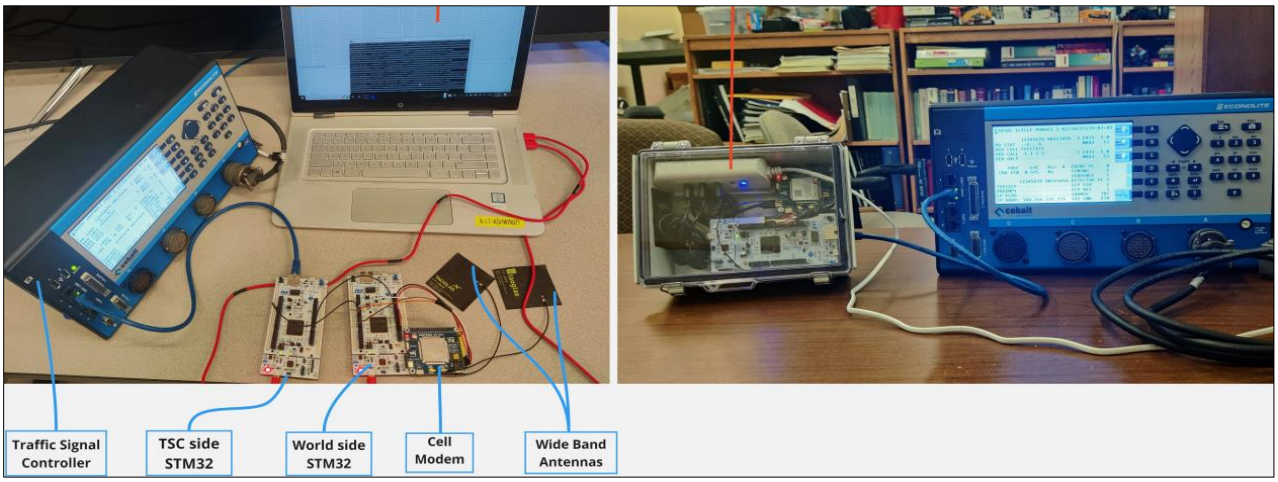


Figure 3.2: Data Diode Practical Representation

CHAPTER 4. COMPONENTS OF THE CODE

4.1 Introduction

The comprehensive suite of software and design resources used in this research 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 in the sections that follow.

4.2 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 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.

4.3. 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, and provides assurance of accurate distribution and appropriate utilization of the data.

4.4 Svelte Mobile App Code

This project uses the Svelte mobile application, 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.

4.5 Python Program for MAP Conversion

The system 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 conducive to efficient storage within the Postgres database, ensuring streamlined data management.

4.6 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.

4.7 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.

4.8 Auxiliary Tools

Complementary resources, such as "tsc simulator.py," and pcap files (can be accessed using Wireshark Software) containing SPaT data have been 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 capabilities 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 5 of this report serves as an informative guide.

CHAPTER 5. DEVICE DEVELOPMENT

The holistic connection framework was introduced in previous chapters. The overarching system can be comprehended through exploration of the various components. The current chapter further elucidates these integral elements to provide a comprehensive grasp of the entire system's workings.

5.1 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 discussed in a previous chapter. The subsequent sub-sections address the intricacies of each of these four sub-tasks, collectively shedding light on their integral roles within the overarching system.

5.1.1 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 made by available through the assistance of Dr. Darcy Bullock of Purdue University. 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 statement of the definitive timeframe for resolution from the McCain Support team.

Consequently, the research team sought to procure another Econolite controller through FHWA's Turner-Fairbank Highway Research Center (TFHRC). This controller emerged as a viable solution for transmitting SPaT data. Through such collaboration, the FHWA TFHRC personnel provided pivotal guidance in programming the Econolite controller. This collective effort not only facilitated the accomplishment of the research objectives.

Programming was carried out to effectively enable the Econolite Traffic Signal Controller (TSC) for transmitting SNMP-based data. This involved a step-by-step sequence, further divided into two distinct sub-parts, as described below:

5.1.1.1 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.

5.1.1.2 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 (Management Information Base) 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 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 illustrated in Figure 5.1, offering a quick-reference overview of the configuration process. Additionally, a sample network configuration for the same is presented in Figure 5.2, providing a tangible representation of how the elements connect within the context of the procedure.

For seamless integration, it is crucial to synchronize the "IPAddress ip" parameter in the TSC Side STM32.ino Arduino IDE code with the specific "SERVER IP" address input during step 5 of 5.1.1.1. More details on Econolite TSC are available in the "Econolite Manuals" folder in github.

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.

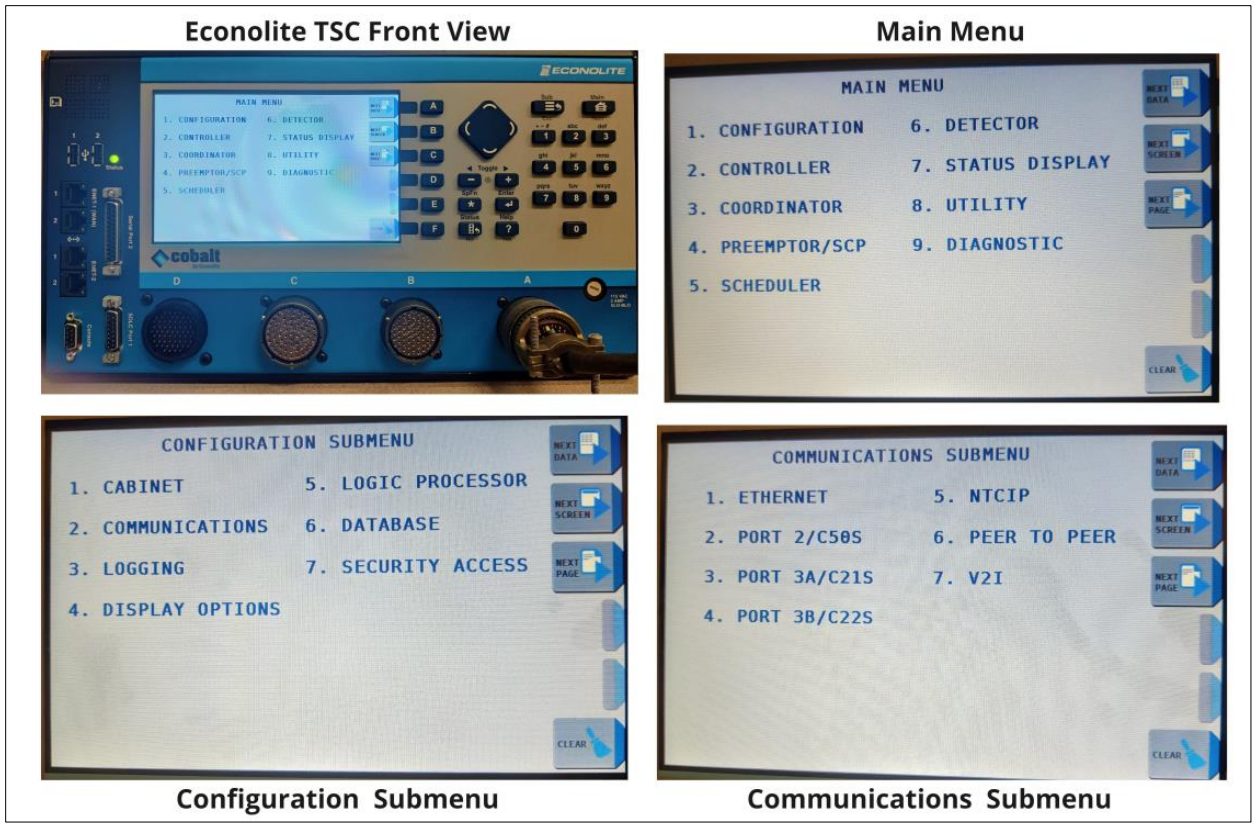


Figure 5.1: Econolite Display Settings

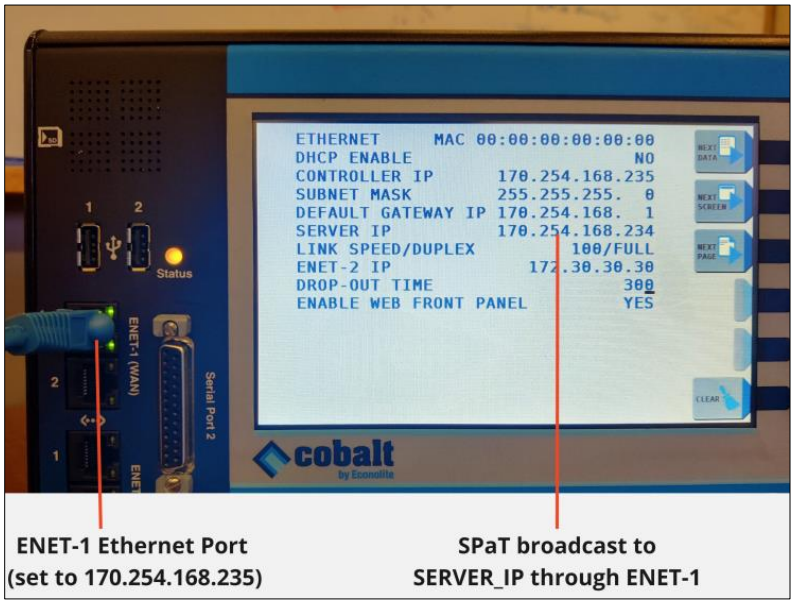


Figure 5.2: Sample Network Settings for Broadcasting on ENET-1

To effectively reconfigure the transmission route, adhere to the following steps:

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

   Here ipAddress is the CONTROLLER IP (i.e. ENET-1 IP) in CONF IGURAT ION → COMMUNICAT IONS → ET HERNET 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 the 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. An 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
```

Figure 5.3: Example interfaces file settings with eth0 (i.e., ENET-2)

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 5.4.

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:04:81:06:62:a0 brd ff:ff:ff:ff:ff:ff
    inet 170.254.168.230/24 scope global eth0:0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:04:81:06:62:a1 brd ff:ff:ff:ff:ff:ff
    inet 169.254.168.235/24 brd 169.254.168.255 scope global eth1
       valid_lft forever preferred_lft forever

$ ip link set dev eth0 up
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:04:81:06:62:a0 brd ff:ff:ff:ff:ff:ff
    inet 170.254.168.230/24 scope global eth0:0
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:04:81:06:62:a1 brd ff:ff:ff:ff:ff:ff
    inet 169.254.168.235/24 brd 169.254.168.255 scope global eth1
       valid_lft forever preferred_lft forever
```

Figure 5.4: Checking of eth0

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 5.5.

```
$ ip route
default via 169.254.168.1 dev eth1
169.254.168.0/24 dev eth1 scope link  src 169.254.168.235

$ ip route add 170.254.168.0/24 dev eth0

$ ip route
default via 169.254.168.1 dev eth1
169.254.168.0/24 dev eth1 scope link  src 169.254.168.235
170.254.168.0/24 dev eth0 scope link  src 170.254.168.230
```

Figure 5.5: Sample ”$ip route” output

7. Set server IP in the communication panel to Diode IP.

8. Push the same SNMP command from 5.1.1.2 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.

However, it is 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 Michigan DOT-configured SERVER IP is not a viable solution, as it has the potential to disrupt their network configuration. In the study, it was then proposed to use traffic mirroring through iptables. The following command was used:

```
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
```

This makes all the data to flow to 169.254.168.234 through ENET-1, mirrored to 170.254.168.234 on ENET-2. Now, 170.254.168.234 can be set 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 is run by the controller, involves a custom built Linux which was not compiled with the feature CONFIG NETFILTER XT TARGET TEE.

This feature is what supports traffic mirroring. It can be noted that that even if one successfully configures the mirroring functionality, there exists a potential risk that the traffic system owner (for example, the 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 utilizing a hub, is shown in Figure 5.6.

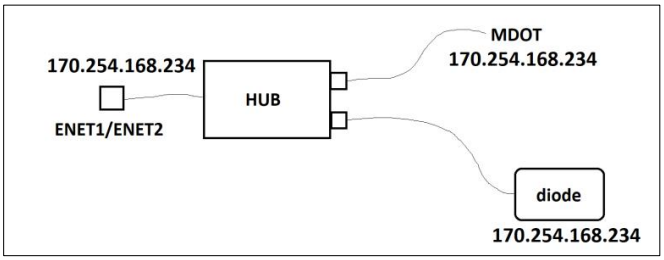


Figure 5.6: Data Broadcasting via Hub as a Potential Solution

Interestingly, during discussions with the traffic system owner (MDOT), a potential resolution emerged. The owner communicated that the ENET2 network was not in active use at the time, and there was a possibility of granting permission to employ the SERVER IP for SPaT data transmission. Given this promising development, detailed exploration of the ipmirroring/hub solution was not aggressively pursued.

When engaging in SSH or sending the snmpset command, it is important to bear in mind that the Ethernet IP address of the PC must reside within the same subnet as that of ENET1/ENET2. It is possible to change the Ethernet IP address in the network settings of the PC. This is shown in Figure 5.7. Further, one could manually set the PC ethernet to the SERVER IP and sniff the SPaT data through Wireshark. This is shown in Figure 5.8.

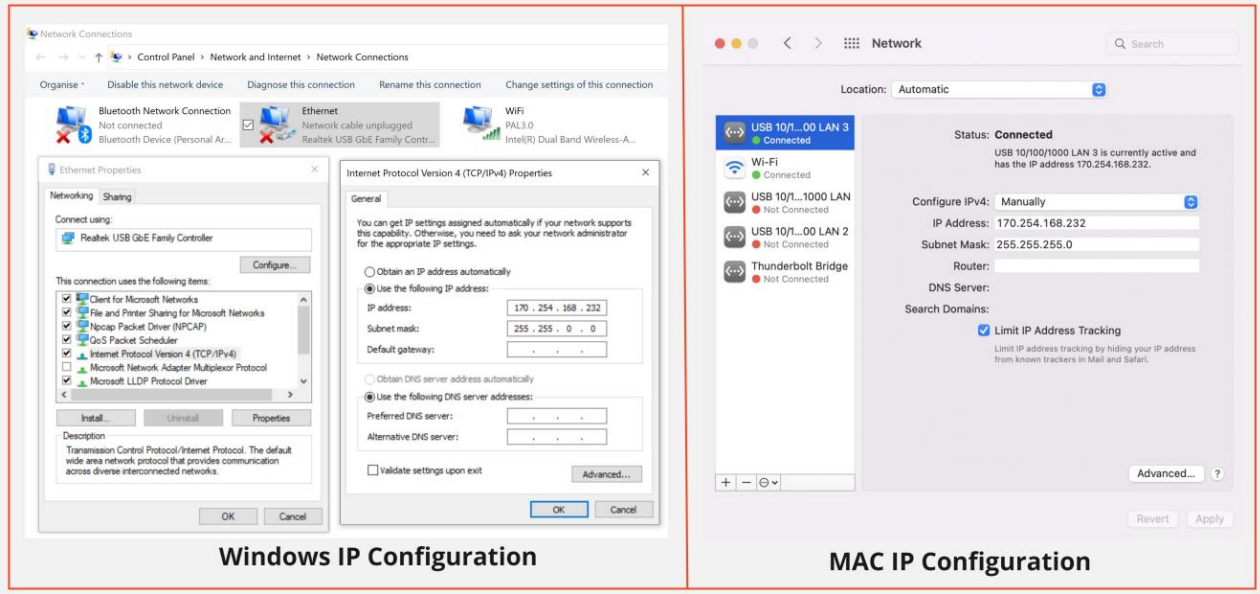


Figure 5.7: Ethernet IP Settings

5.1.2 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 Controllerside and the World side components [4]. 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. Additionally, the competitive price point ensured cost-effectiveness without compromising functionality.

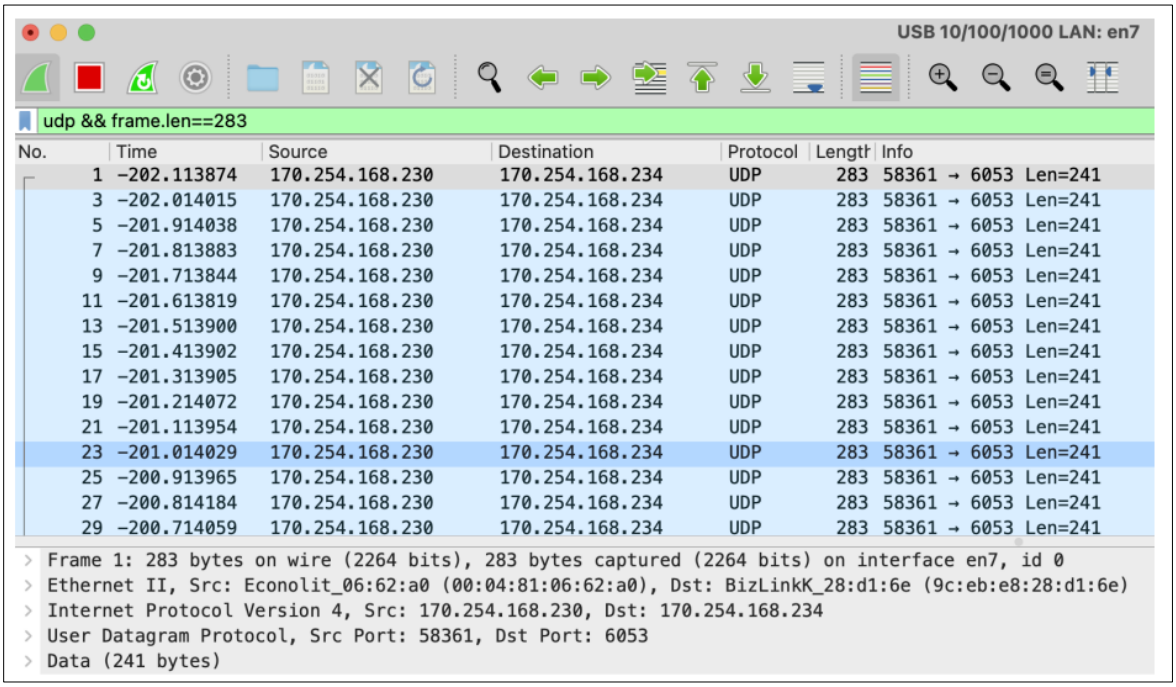


Figure 5.8: Data Sniffed and Filtered in Wireshark
(Note that the PC Ethernet is set to SERVER IP)

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

Appendix A.3 of this 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.

The following procedure is followed to set up Arduino IDE:

1. In Arduino IDE install "STM32duino STM32Ethernet" library from *Tools → ManageLibraries → 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 5.9, Figure 5.10 and Figure 5.11.

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 chapter 4 under the path ”DATA DIODE\Data Diode efficient heap based\TSC Side STM32 can be uploaded after setting the ip address, as follows:

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

This object in the file needs to be same as the server address set in the TSC as explained in Subsection 5.1.1 of this report. 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 could be modified 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 could be checked by sniffing the data through wireshark. Also, it is 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. Such blinking serves as an indication of the successful transmission of data over the network connection.

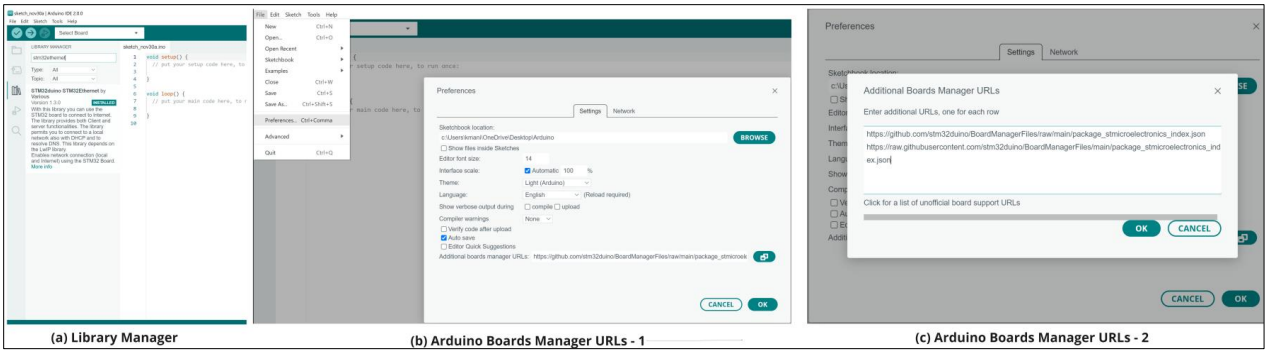


Figure 5.9: Arduino Library and Boards Manager URLs

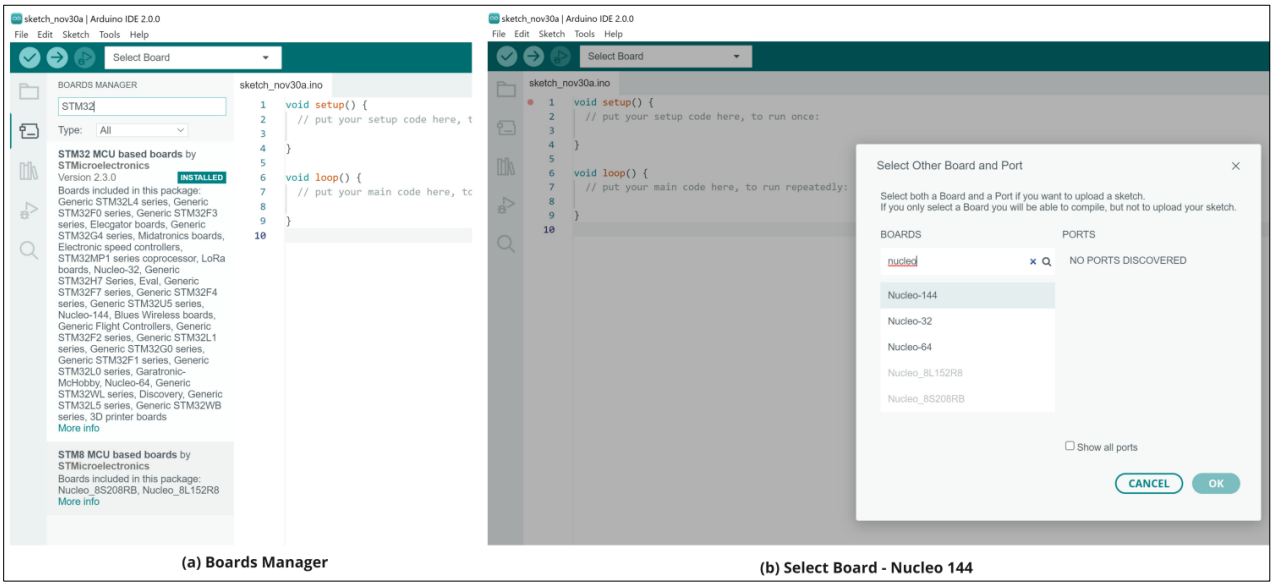


Figure 5.10: Arduino Boards Manager

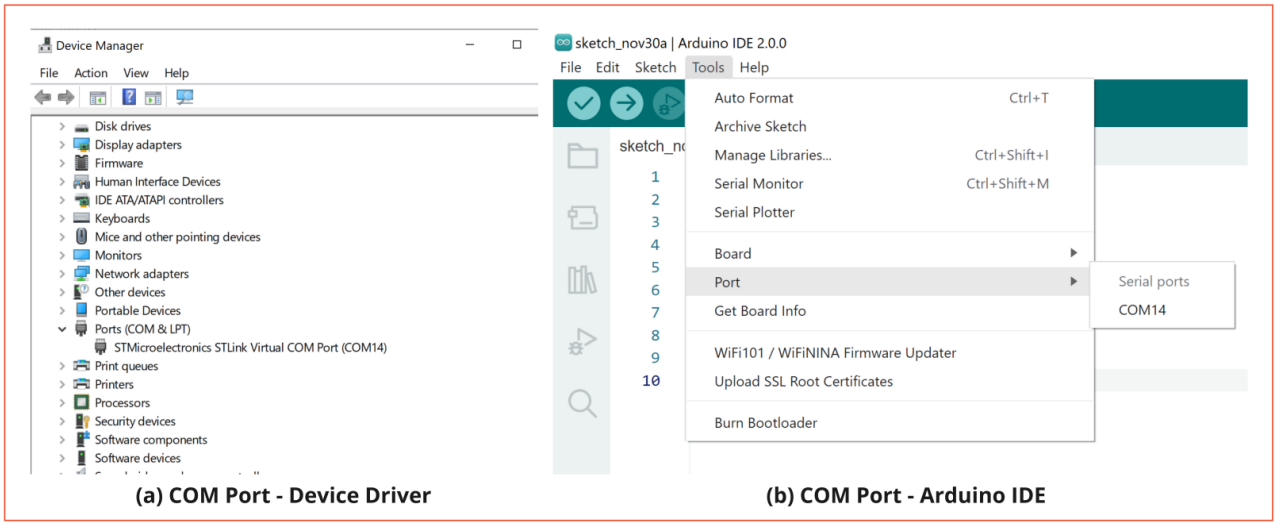


Figure 5.11: COM Port View

5.1.3 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 SPaT data (from TSC). 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. Also, after 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 often includes 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 in the context of the system's communication requirements.

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

After the programmed STM32 (described in Subsection 5.1.2) is connected to the configured TSC (described in Subsection 5.1.1), one should be able to see the serial prints in the Arduino IDE Serial Monitor. Figure 5.12 presents a sample view of such serial prints. 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) [5] 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 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 5.12.

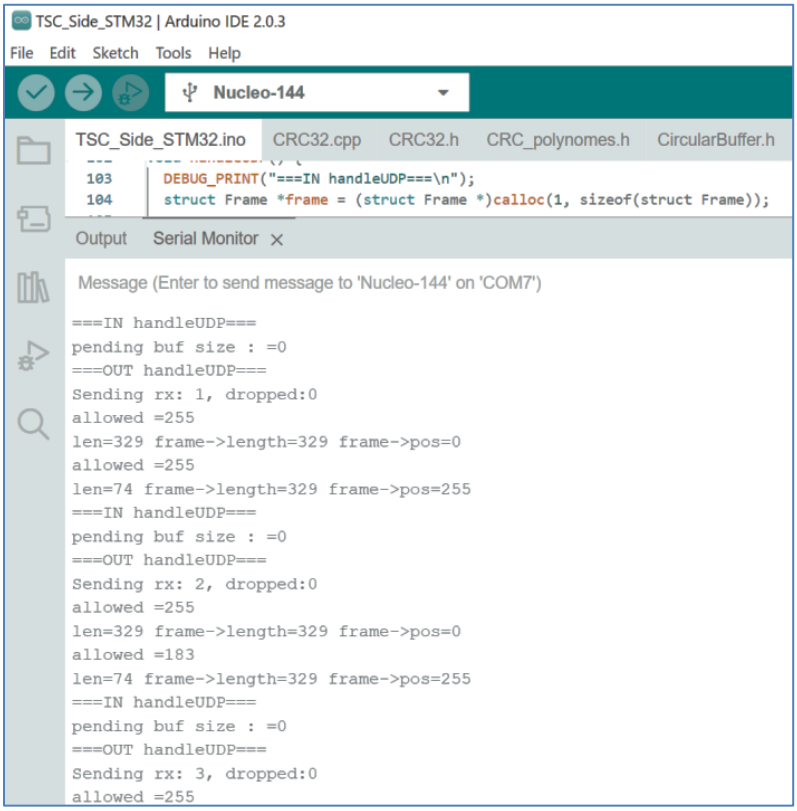


Figure 5.12: Minimal Serial Print Logs from TSC Side STM32.ino File

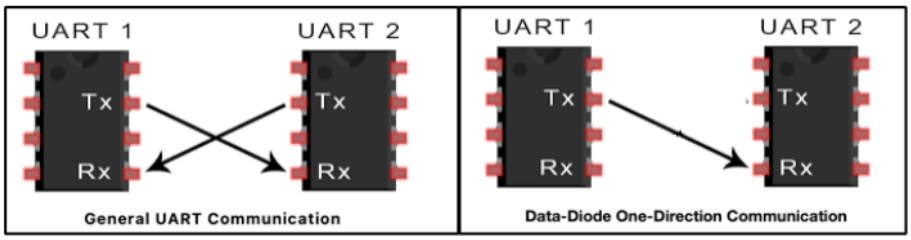


Figure 5.13: Data Diode One-Direction Communication

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. This is a fundamental characteristic of the Data Diode's operation: the TSC is isolated from potential spurious attacks (Figure 5.14).

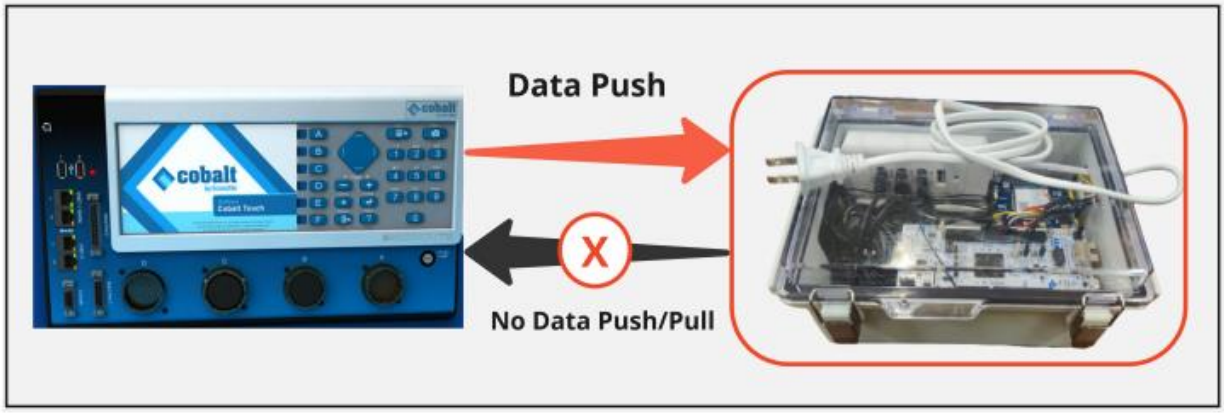


Figure 5.14: TSC-Diode Data Push Mechanism

A Baud rate of 115,200 was used for the demonstration. The programmer must take care of setting the same Baud rate between the two STM32s. The Tx Ports and 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 5.15. 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 5.16, while the cmd window output of "com stm.py" is shown in Figure 5.17. (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 could 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 5.18. It is of paramount importance to comment or remove these data print statements, as they could potentially consume a significant amount of CPU cycles.

5.2 The World side of the Diode

The World side of the Data Diode is realized by executing 3 sub tasks as mentioned in an earlier chapter. Each of these 3 sub tasks is explained in the following sub sections.

5.2.1 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 microcontroller.

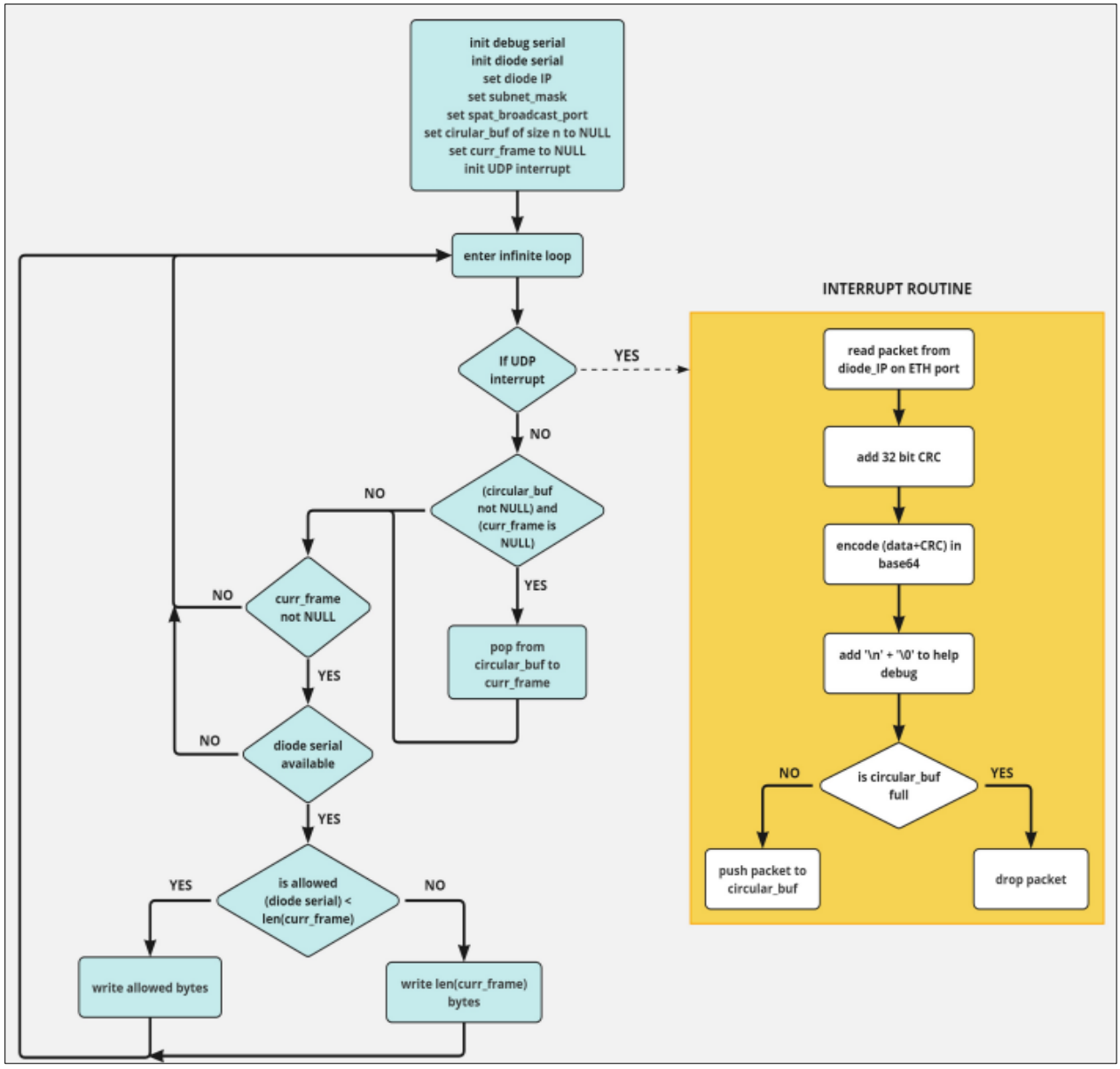


Figure 5.15: TSC Side STM32 Codeflow

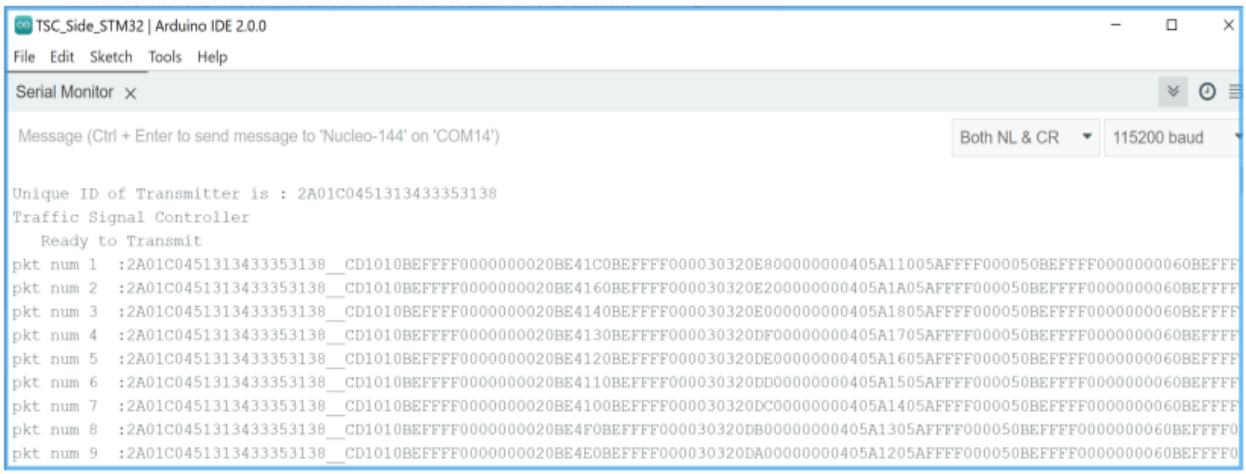


Figure 5.16: Serial Print Logs from alternate TSC Side STM32.ino File

```
C:\Users\kmani\OneDrive\Desktop\Data_Diode_Report\Data_Diode_SW_Code>PYTHON com_stm.py
Unique ID of Transmitter is : 320330165131343335 3138

Traffic Signal Controller

   Ready to Transmit

pkt num 1  :CD1010BEFFFF0000000020320F9032FFFF0000305AFFFF00000000405A2A705AFFFF000050BEFFFF0000000060320F9032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
0000100000000000000FFDD00022FFFF0000FFFF000000000084E0E31300

pkt num 2  :CD1010BEFFFF0000000020320F8032FFFF0000305AFFFF00000000405A2A605AFFFF000050BEFFFF0000000060320F8032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
0000100000000000000FFDD00022FFFF0000FFFF000000000084F0E31300

pkt num 3  :CD1010BEFFFF0000000020320F7032FFFF0000305AFFFF00000000405A2A505AFFFF000050BEFFFF0000000060320F7032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
0000100000000000000FFDD00022FFFF0000FFFF000000000008500E31300

pkt num 4  :CD1010BEFFFF0000000020320F6032FFFF0000305AFFFF00000000405A2A405AFFFF000050BEFFFF0000000060320F6032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
0000100000000000000FFDD00022FFFF0000FFFF000000000008510E31300

pkt num 5  :CD1010BEFFFF0000000020320F5032FFFF0000305AFFFF00000000405A2A305AFFFF000050BEFFFF0000000060320F5032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
0000100000000000000FFDD00022FFFF0000FFFF000000000008520E31400

pkt num 6  :CD1010BEFFFF0000000020320F4032FFFF0000305AFFFF00000000405A2A205AFFFF000050BEFFFF0000000060320F4032FFFF000070
5AFFFF00000000805AFFFF05AFFFF0000900000000000A000000000000B000000000000C000000000000D000000000000E000000000000F00000000
```

Figure 5.17: CMD Window Output of ”com stm.py”

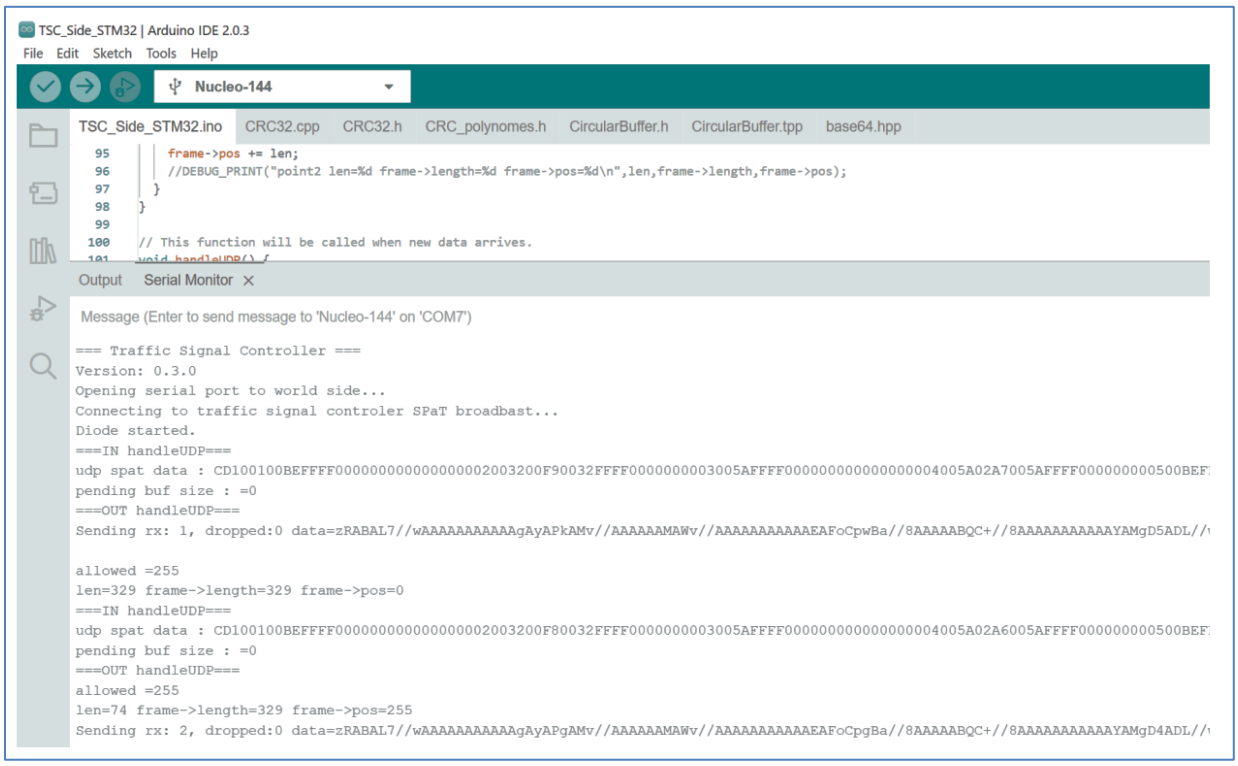


Figure 5.18: SPaT and base64 Serial Print Logs from TSC Side STM32.ino File

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 is worth noting that, similar to the STM32 on the TSC side, these traffic loggings could be disabled on both STM32 microcontrollers by undefining ”DEBUG” parameter in the file. Such flexibility in disabling the logging could help conserve CPU processing cycles and optimize the system performance. Figure 5.18 presents a sample view of the serial data traffic and prints.

During the operational phase, the serial print of the ”data” parameter in the line ”New message. Length =” which represents the incoming data, is deliberately omitted from being printed. Such 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 instead of printing debugging information to the serial monitor. To ensure proper communication, it is essential for the programmer to set the same Baud rate for communication between the two STM32 microcontrollers. This synchronization of Baud rates ensures that the data can be accurately transmitted and received between the components of the diode, thereby facilitating smooth operation and reliable data exchange.

5.2.2 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 modem manual were used 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 5.22 , it can be seen 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 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 is essential to disconnect the jumper between the 3.3V and PWR pins of the modem, particularly where 5V is provided as input. After 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 is 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 has to be placed at 90-degree angle with respect to the other. The datasheet of the antenna is referenced in Appendix A.4.

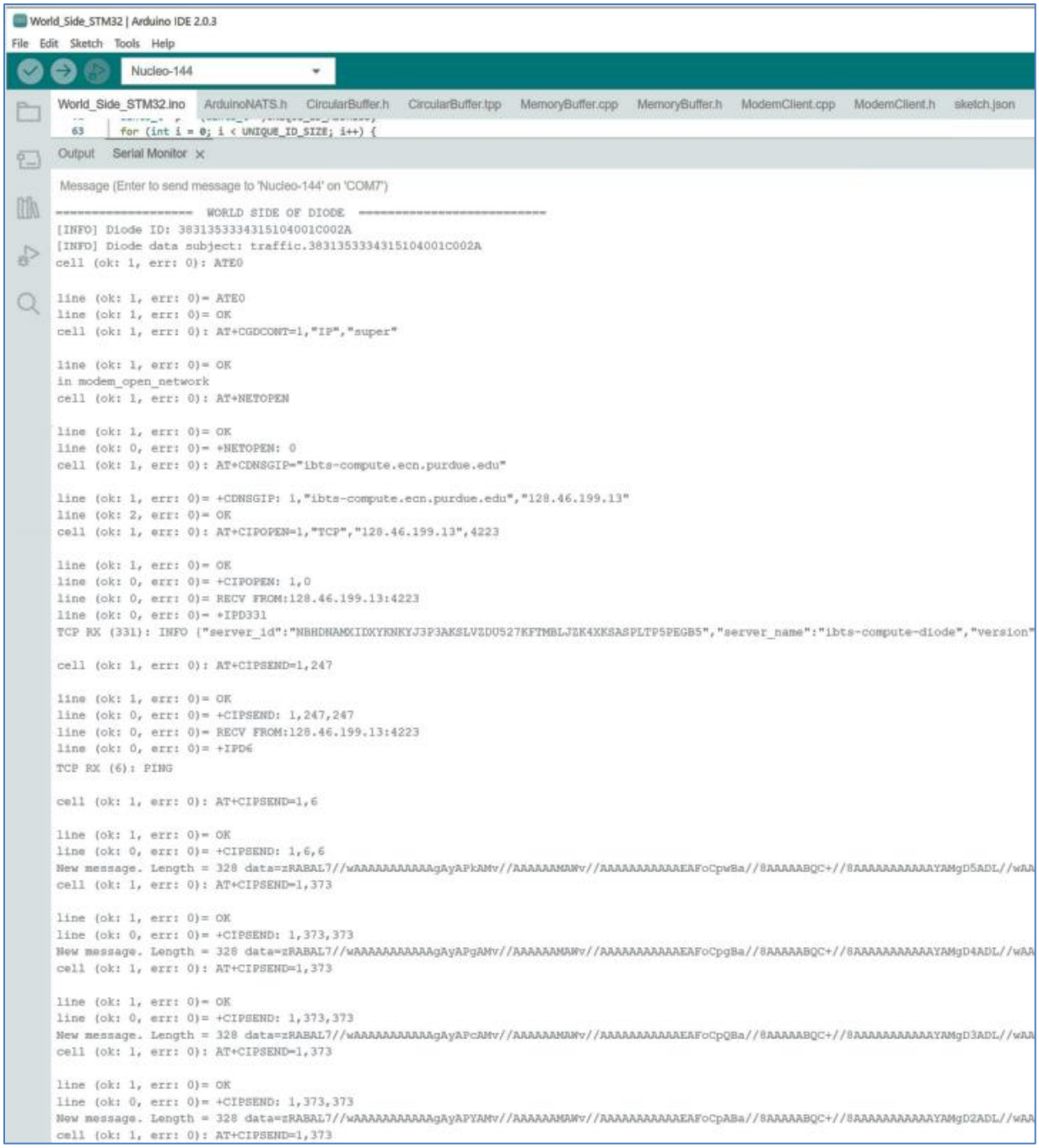


Figure 5.19: Serial Print Logs of World Side STM32.ino File

In addition, it is essential to place the antennas appropriately. 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, the reader is requested to 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 used. This is 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. Figure 5.20 presents the State machine diagram outlining the different states of the modem client.
This modem client is responsible for several tasks. The most important of these tasks 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.

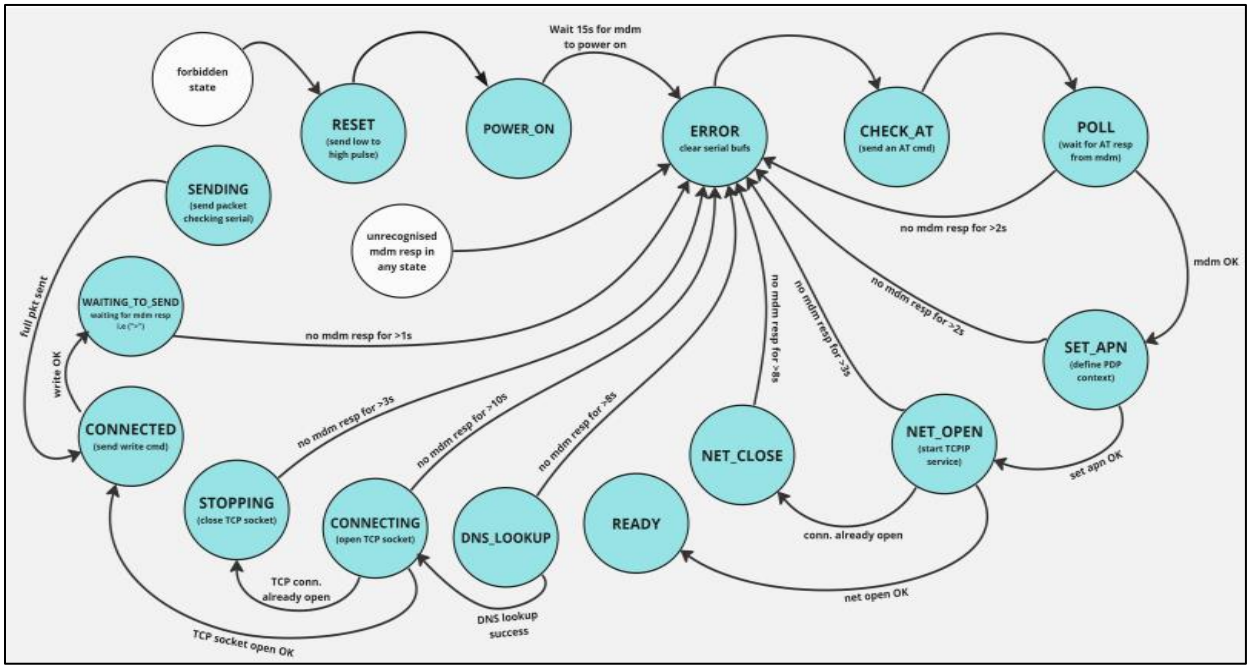


Figure 5.20: Modem Client State Machine

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 the example has the NATS address set to ibts-compute.ecn.purdue.edu and the Server listened 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.

5.2.3 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 are 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 available in Chapter 45 (Device electronic signature) of the STM32 reference manual. The manual is cited in Appendix A.3 of the report.

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, 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 indicated when the LED3 light turns 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 5.21 and the interconnection of the components described thus far is shown in Figure 5.22.

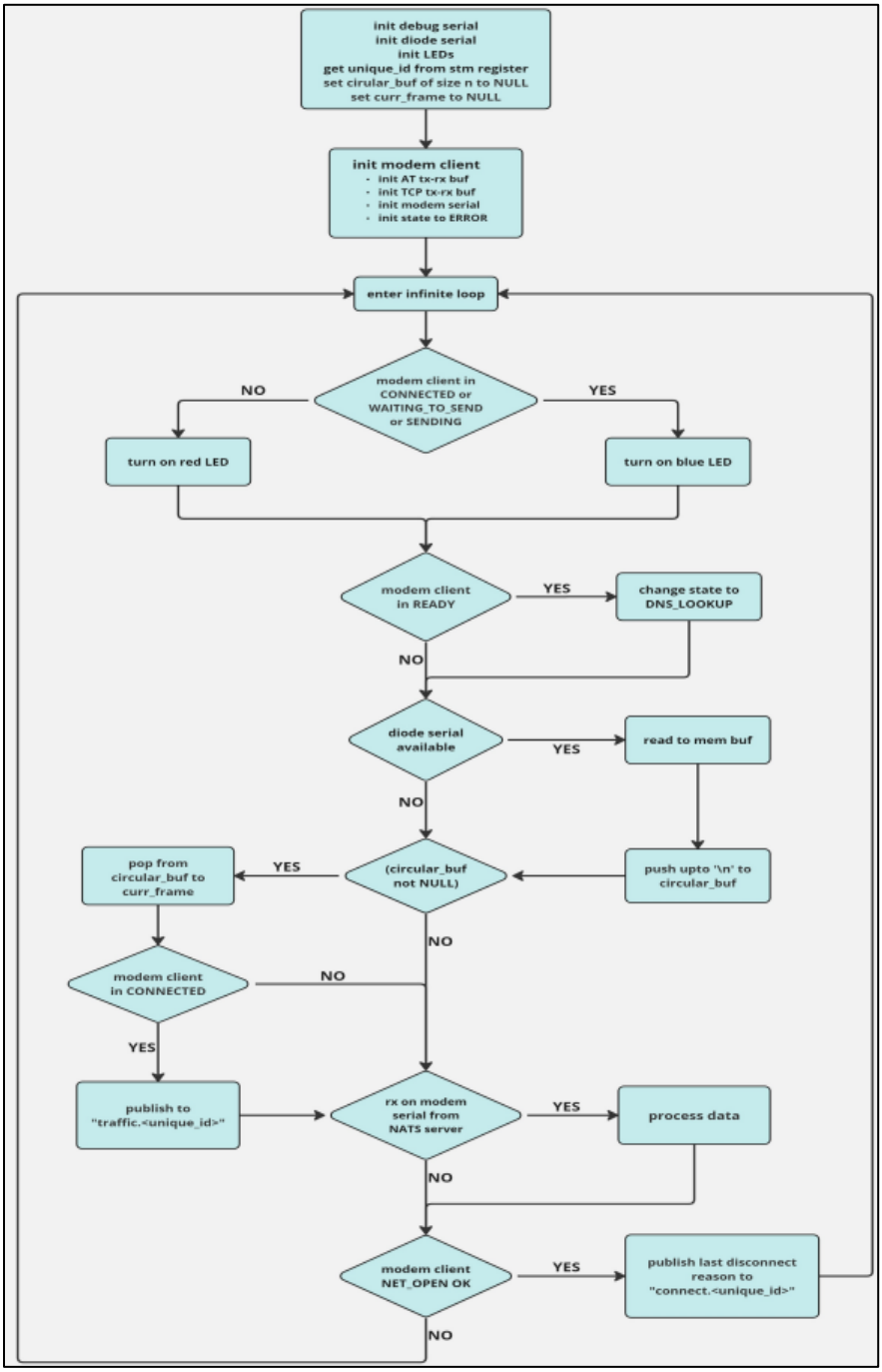


Figure 5.21: World Side STM32 Codeflow

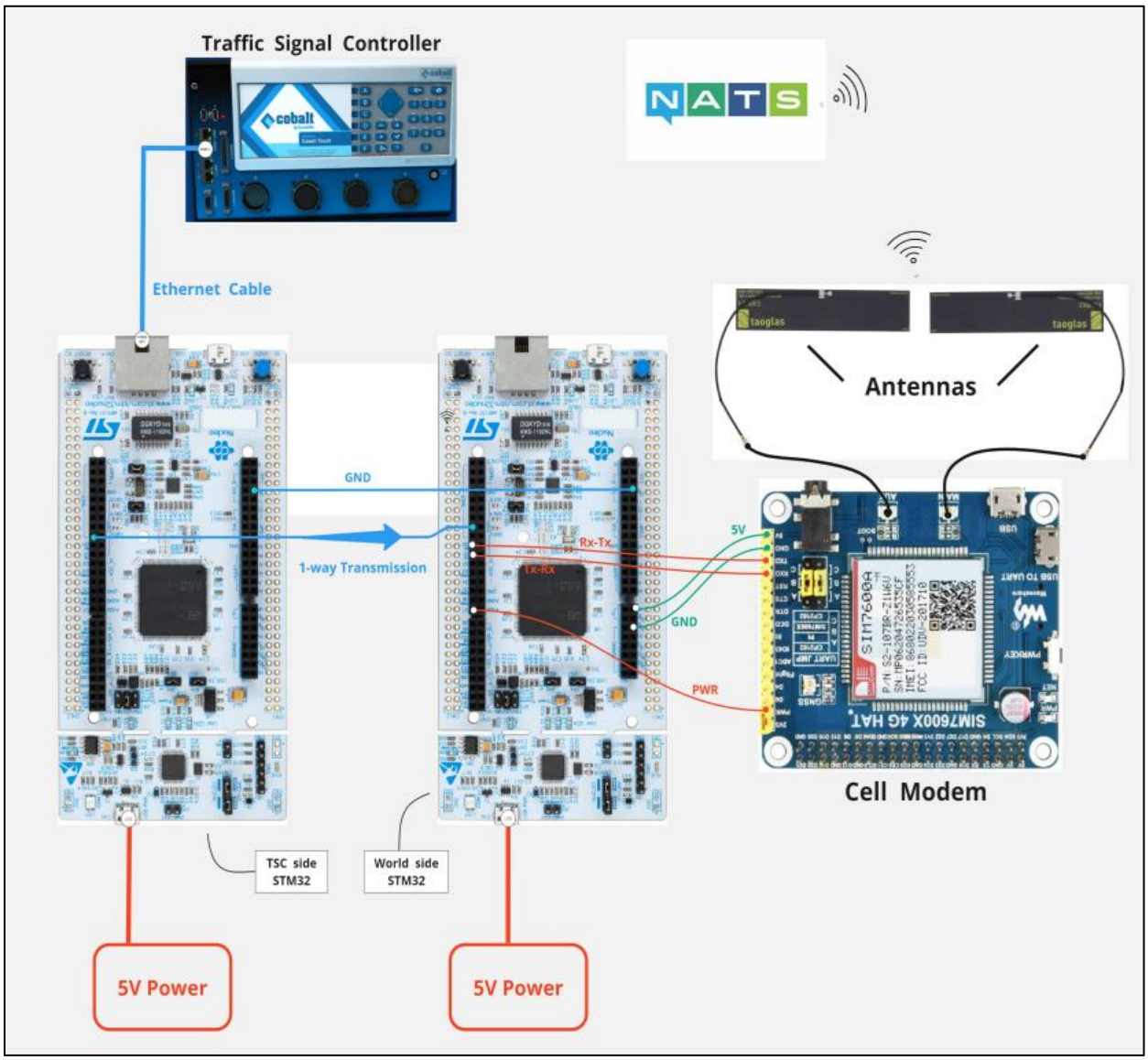


Figure 5.22: Data Diode Components Interconnection

Similar to the STM32 on the TSC side, if it is needed to capture data in transit through the STM32 on the World side and to save it in a file or view it on the serial monitor, there exists the option of using a separate ”World Side STM32.ino” file located in git repo of chapter 4 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 wishes to log the serial data into a text file, one could use the same ”com stm.py” script as mentioned before. The Serial prints generated by this code version are presented in Figure 5.23.

It is 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 5.23: STM32 Serial Traffic Prints on World side of the Diode from alternate ”World Side STM32.ino”

It may be noted that the COM Ports of the two STM32s will be different and if code upload is done using the same PC (which typically, is the case), care must be taken to choose the appropriate COM port before giving the upload command. Also, it may be noted 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 software packages like Putty or Tera-Term. This scenario is depicted in Figure 5.24. 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.

5.2.4 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 blinks 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.

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. Figure 5.24 presents the Nucleo Board with the labelled LEDs.

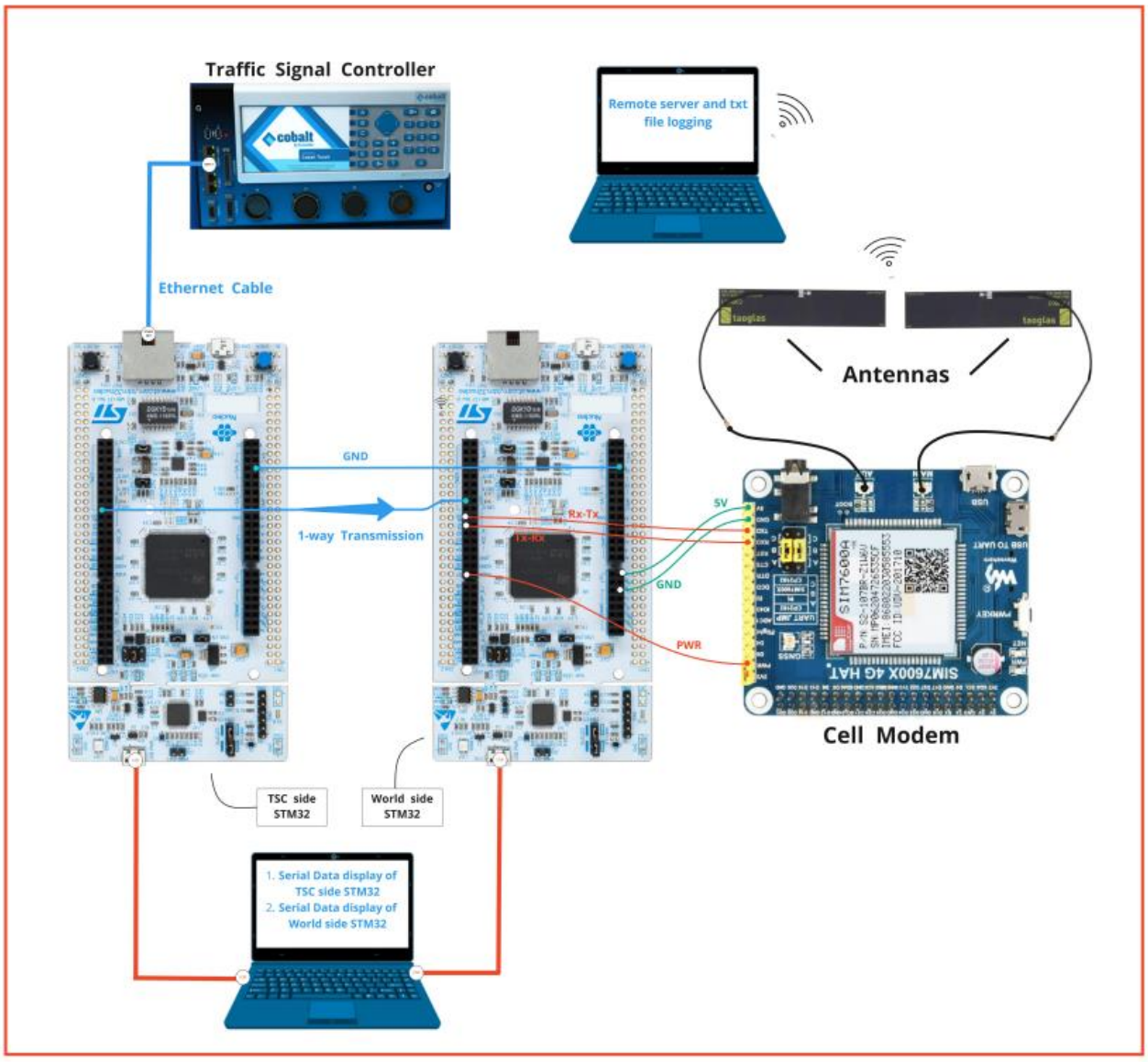


Figure 5.24: Serial Data Logging

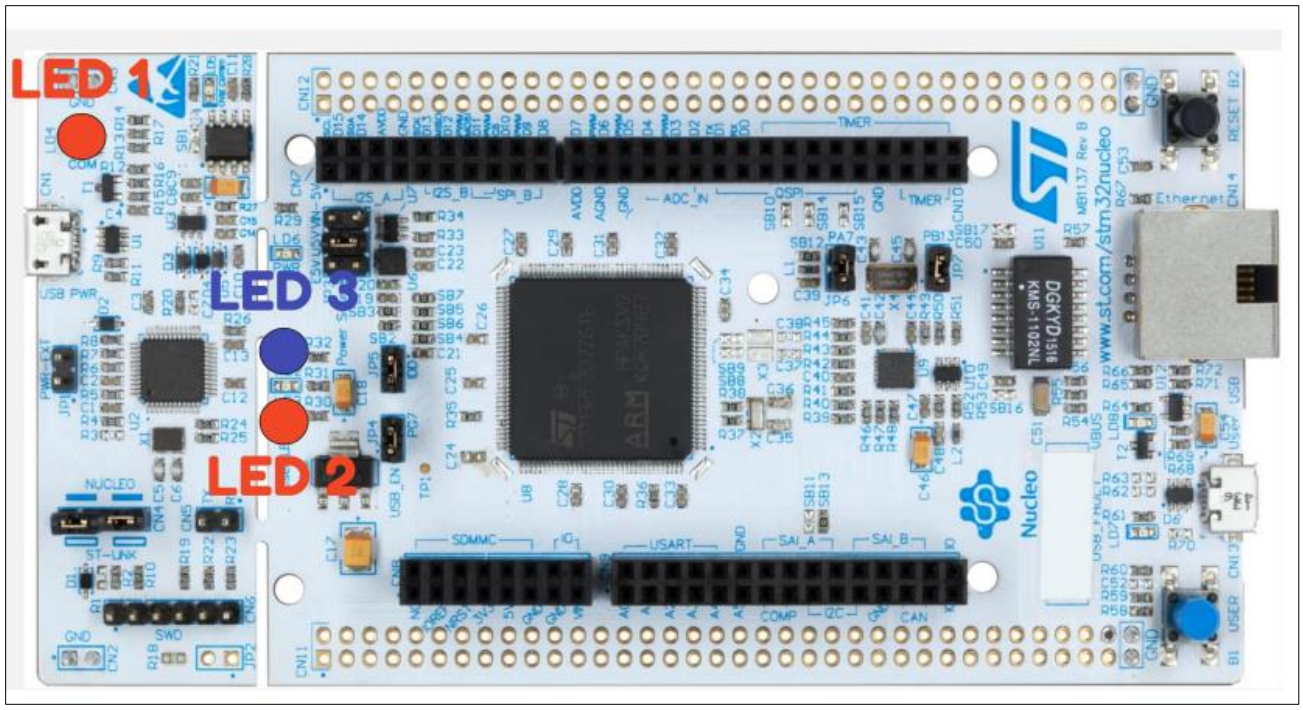


Figure 5.25: World Side STM32 LED Indicators

5.2.5 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
```

Figure 5.26: Optimizing Serial Buffer Size

The location of this file may vary based on the system. The system used for development during this project demonstration 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 are 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 is 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 could lead to the occasional occurrence of a race condition, causing unpredictable behavior in the Serial communication. Therefore, when expanding buffer sizes beyond 256, it is essential to exercise caution by closely monitoring the system's behavior.

5.3 TSC Simulator

When a TSC is available, the ”tsc simulator.py” in the git repository of chapter 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. A few PCAP files are already made available in the ”pcap files” directory in the same path. The user could input these files by modifying ”tsc simulator.py” accordingly. A few points to be noted while running the ”tsc simulator.py,” are:

1. The ”localPort” value on ”TSC Side STM32.ino” file must match 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, in turn, 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 several 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 are provided in Chapter 5.5.4 of the current report.

Figure 5.27 presents the system interconnection while replaying the pcap data onto the Diode using the ”tsc simulator.py” and Figure 5.28 shows the method to run ”tsc simulator.py” and its sample output.

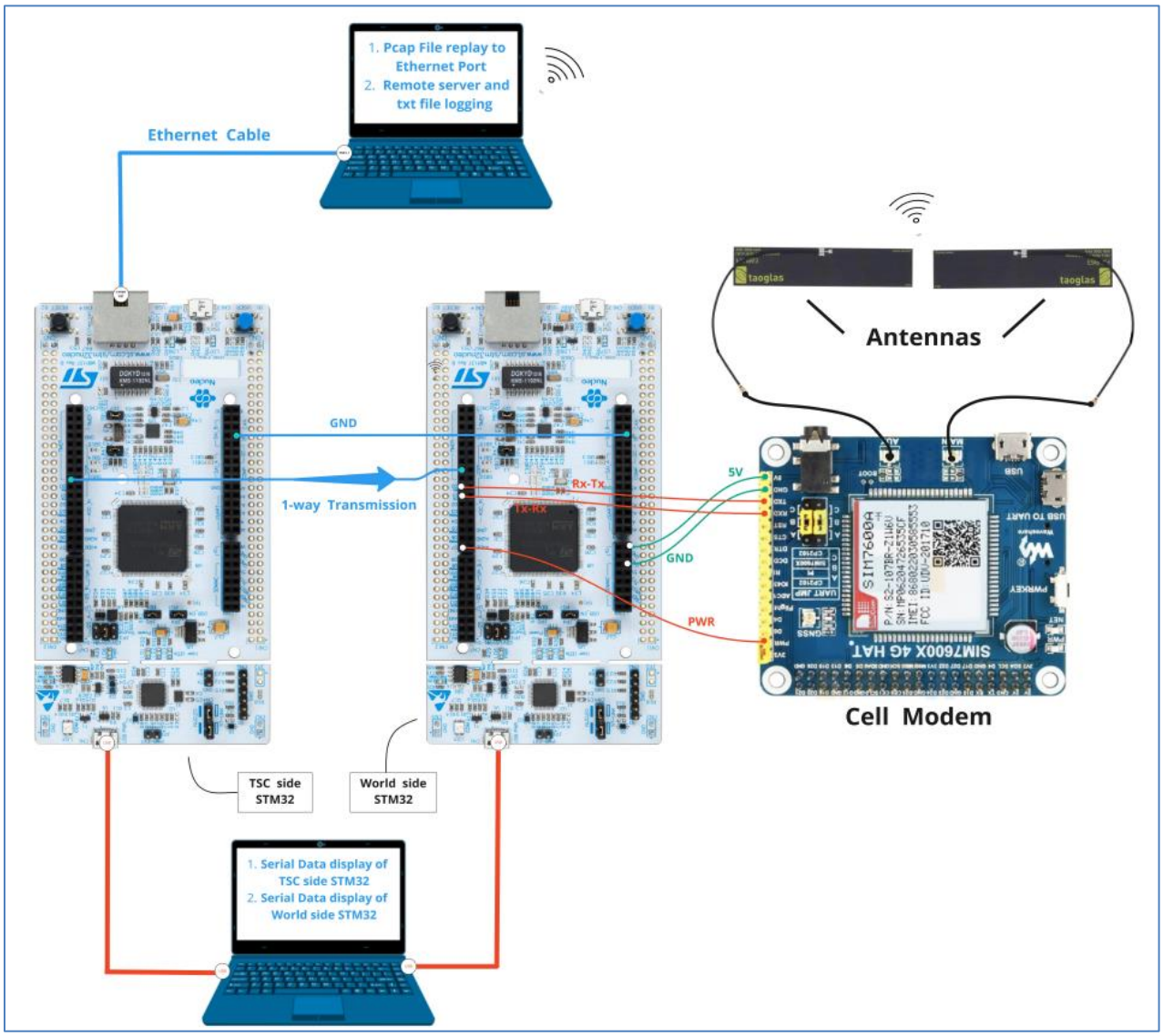


Figure 5.27: Simulating a TSC by replaying PCAP file to the PC Ethernet Port

5.4 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 [3].

NATS uses 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 are then exchanged.

The decision to adopt NATS was guided by its several benefits. First, NATS offers a secure, ultra-low latency, and high-performance data transfer mechanism. Also, it has ease of implementation, comprehensive documentation, and support for NATS client in multiple programming languages. Further, 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 possess direct knowledge of each other's presence. Instead, they communicate through subjects, eliminating the need for specific point-to-point connections. This design stands in contrast to conventional Remote Procedure Call (RPC) technologies that require a dedicated one-to-one connection between the requester and the replier.

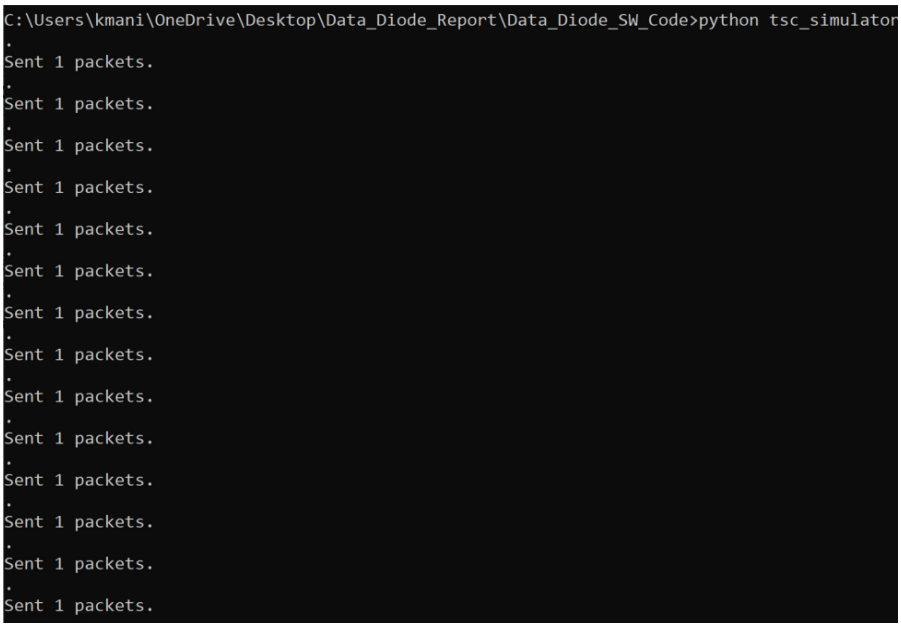


Figure 5.28: Sample output of tsc simulator.py

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. Further information on NATS and its capabilities is provided at the official NATS website at: https://nats.io/.

5.4.1 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 diode. 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.

5.4.2 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 one to replicate and assess different scenarios without the need for physical devices or deployments.

With the NATS client, one could 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, one could emulate position and light status updates that would typically be received from vehicles or devices near intersections.
By setting up this testing environment, one could evaluate how your system handles different data inputs, interactions, and potential challenges. This approach allows one to finetune and optimize one’s 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”

One has the flexibility to utilize the NATS demo server as part of one’s 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 current project, a specific server was established at the Agricultural & Biological Engineering (A&BE) department at 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 port4223. This server is maintained and managed by the OATSCenter (Open Ag Technologies and Systems Center) at A&BE.

5.4.2.1 Emulating a diode

If a server is already setup, then telnet to the server will help to establish a client connection with the server. Otherwise, a local server could be run to continuously listen to the client. Figure 5.29 presents an instance of client-server messaging setup on a MAC terminal.

If a server is already set up and running, one could 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 could 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. Figure 5.29 presents an example of client-server messaging setup on a MAC terminal.

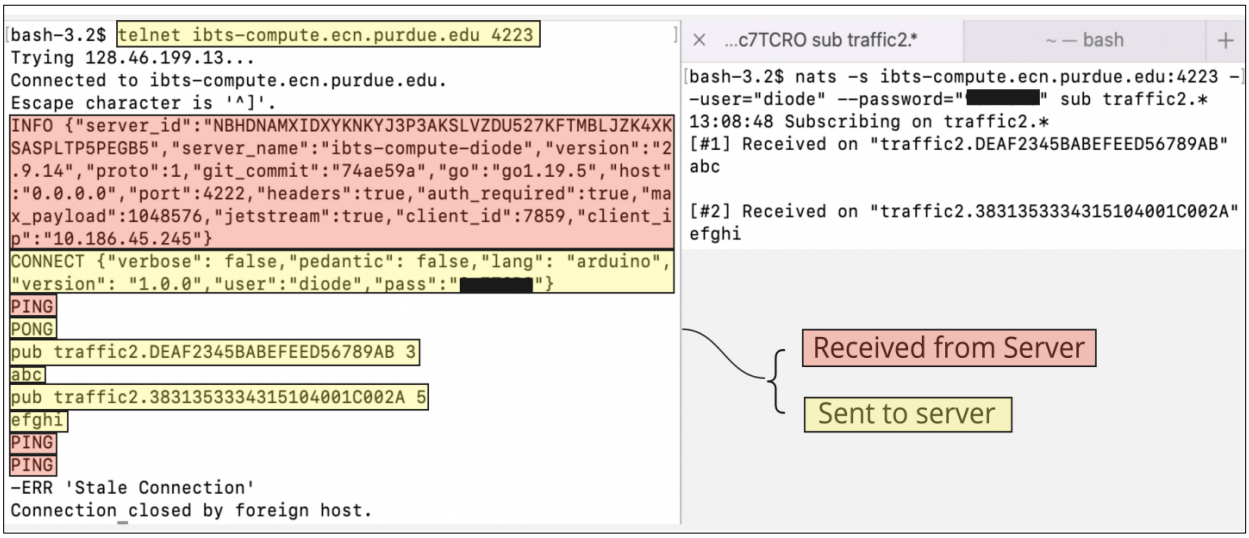


Figure 5.29: Emulating Diode messages with NATS CLI tool

Figure 5.30 presents 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.

5.4.2.2 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” script to the web app is provided in README file in the ”mobile” section of the github repository. Figures 5.31 and 5.32 present the commands in action on the MAC terminal.

Figure 5.30: Using demo.nats.io with NATS CLI tool

Figure 5.31: Subscribing to the subject traffic.*

```
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 5.32: Emulating ”light-nearest” reply

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

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 has not 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 is important to note that while a CRC check can help detect some types of data corruption, it is not foolproof. There exists 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.

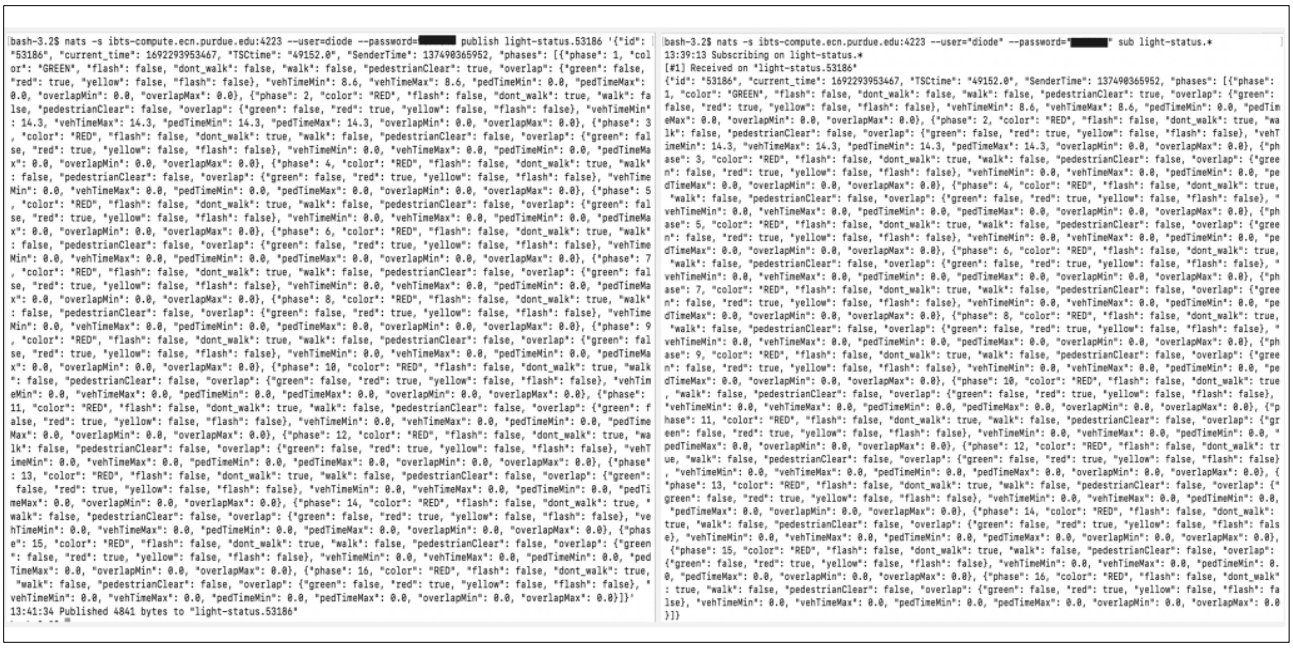


Figure 5.33: Emulating RED Light Status

In each phase, two parameters ”vehTimeMin” and ”vehTimeMax” are 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, one could refer to Figure 5.34 in the supporting 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. These documents provide insight into details 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 5.34: Traffic Signal Controller Broadcast Message Version #2 Byte-Map Structure

5.4.4 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 does not 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 example, 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 are directed to the appropriate intersection subject for further processing.

The 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 subject 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 are parsed as per NTCIP protocol. The parsed data are 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 are made available for subscribers 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 5.35.

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

Figure 5.35: Re-published SPaT JSON Message Unit

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 could be seamlessly understood and utilized.

To facilitate the creation of the MAP data unit, a specialized tool was used. This tool can be accessed at the https://webapp.connectedvcs.com/ called the Intersection Situation Data (ISD) Message Creator. The 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.

The step-by-step procedure for creating a MAP data unit is provided below:

1. Identify Ingress Lanes : Start by identifying all lanes that approach the intersection. These are known as ingress lanes. It is 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 could 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 are provided subsequently in this report.

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. The Virginia Tech team collected and organized the intersection information using the ISD Message Creator tool and constructed the MAP data unit for each intersection. Figure 5.36 presents a snippet from the ISD Message Creator tool used for creating MAP data unit.

The MAP files containing information on 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.

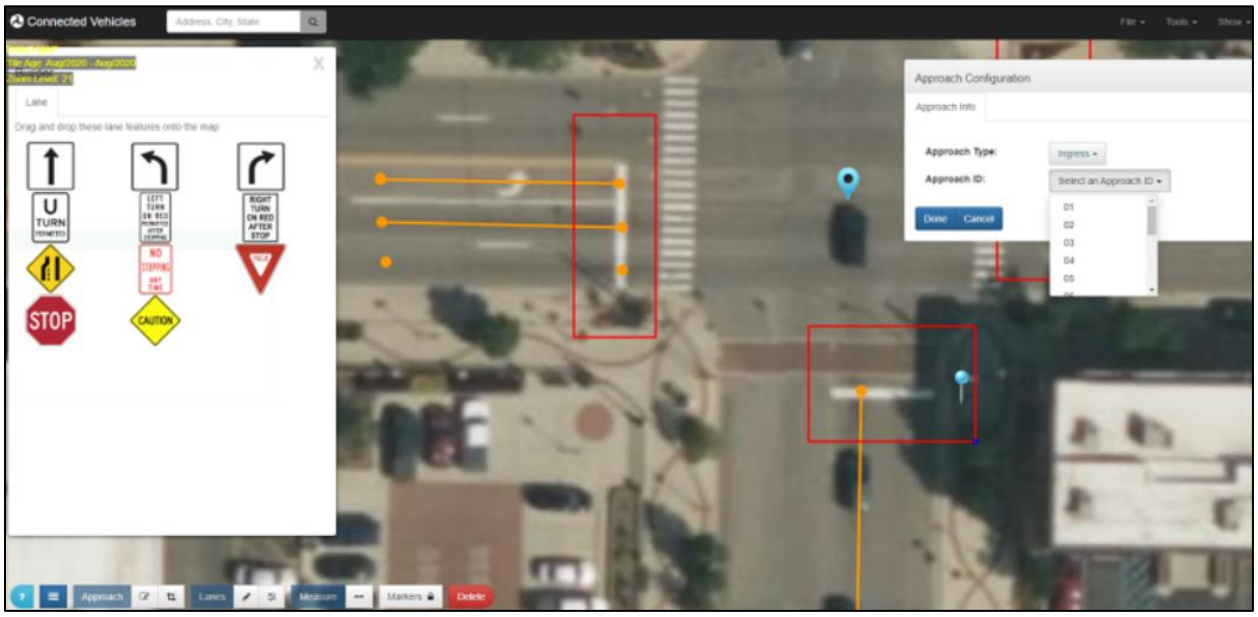


Figure 5.36: ISD Message Creator Tool Sample View

5.4.5 Responding to the app's queries 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 are 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. Figure 5.37 presents the cmd output of this ”map to postgress.py” script, and Figure 5.38 presents the ”lanes” table created in Postgres.

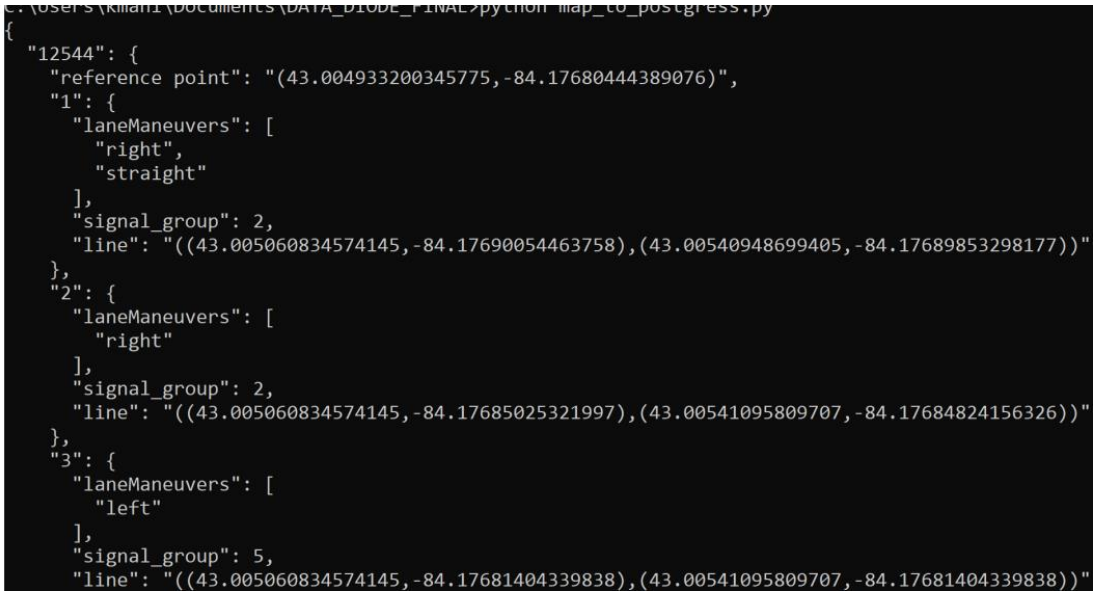


Figure 5.37: ”map to postgress.py” script output

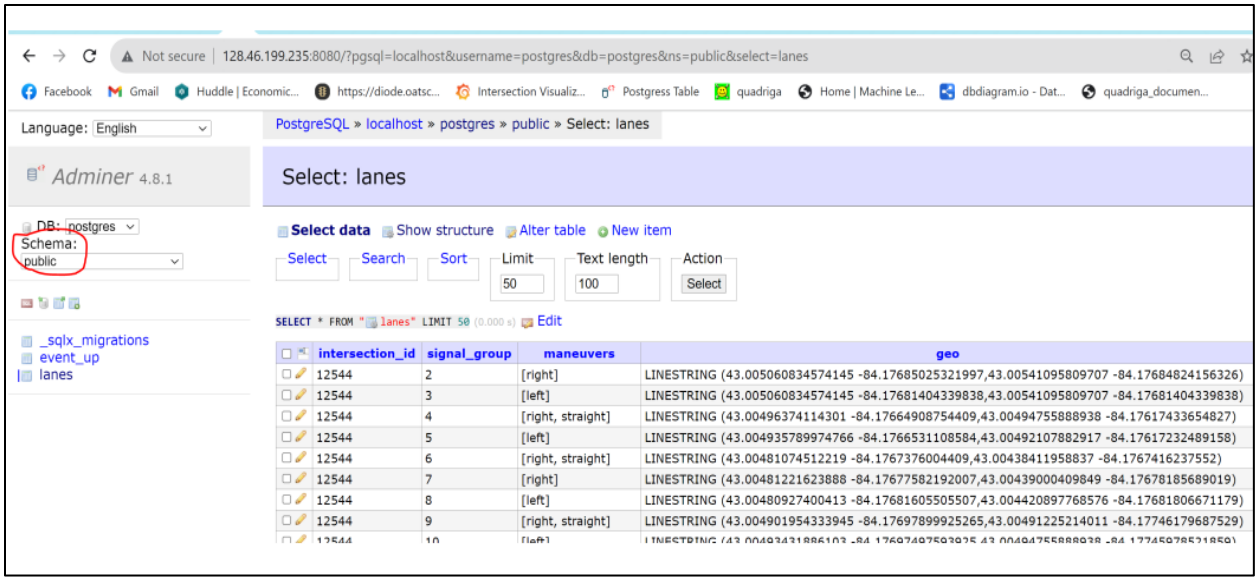


Figure 5.38: Lanes Table in Postgres

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 of the nearest lane, the corresponding signal group, maneuvers allowed in that lane, and the intersection ID of the queried GPS coordinate.

Using 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 discern 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 5.39. 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 5.40. The Backend Script in this figure is ”nats parsing.py”.

5.5 Web App Side of the System

The 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 user interface 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. Its 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 one could use as a foundation for building one's app. These components cover various aspects of UI, including forms, navigation, layout, and more. Additional information could be obtained at https://kit.svelte.dev/

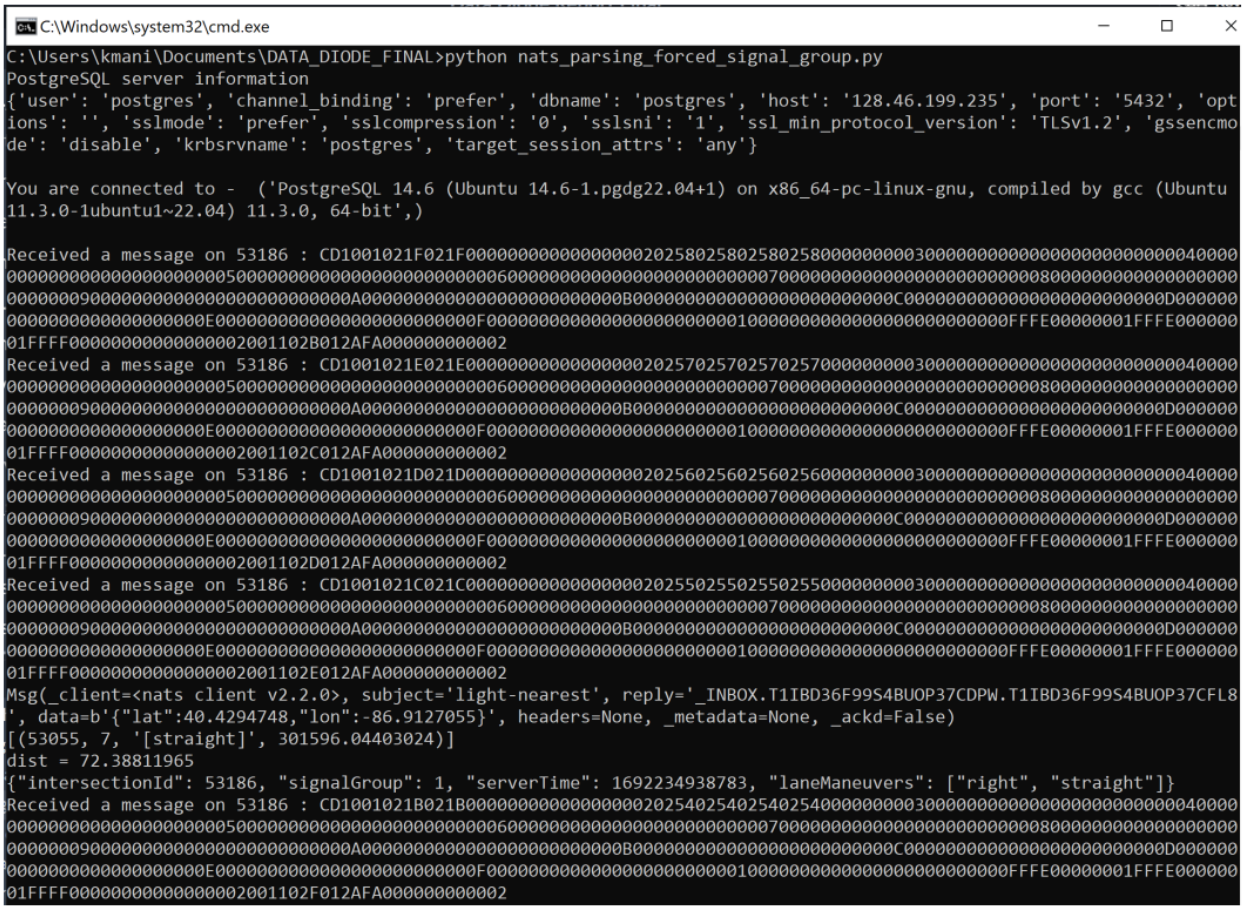


Figure 5.39: "nats parsing.py" script output

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

The **watchPosition()** method of navigator.geolocation plugin is used to register a handler function. This function is called automatically any time the device position changes. Since there was minimal attention to build the app as it was out of scope of this project, attention was not given to accuracy of the gps or the refresh rate of the gps. The direction info from the gps could 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. Higher accuracy and more frequent updates may result in better location tracking but could also consume more battery and data.

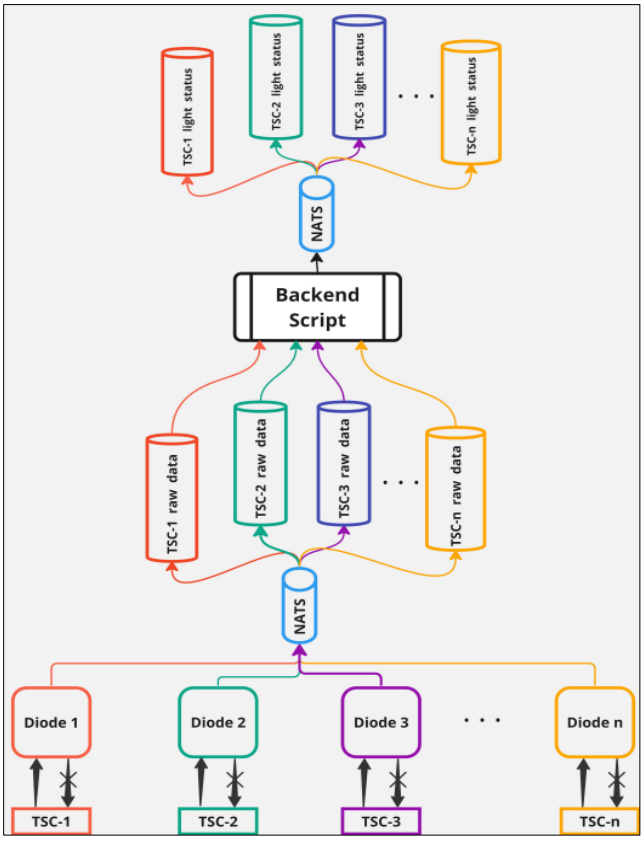


Figure 5.40: NATS Data Routing Mechanism

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

It is 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 could affect GPS accuracy.

5.5.2 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 is important to note that the accuracy of the displayed information depends on the accuracy of the GPS coordinates received from the device. Variability in GPS accuracy could lead to variations in the precision of lane and intersection information displayed by the app.

5.5.3 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.

5.5.4 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 is essential to consider the latency introduced by the communication process. In the reference to the current project, the latency could 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 used 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 1,000 packets, and the latencies from these packets were averaged to derive the average latency. Additionally, the minimum and maximum latencies among the 1,000 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 used. 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 Purdue University's Materials and Electrical Engineering department (Figure 5.41):

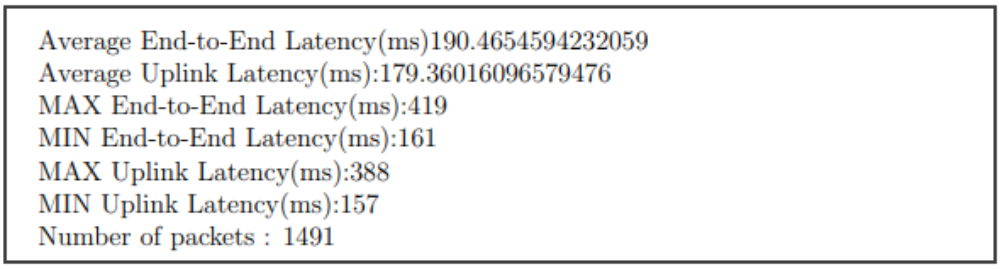


Figure 5.41: Latency measured using the app located at Purdue's Materials Science & Electrical Engineering Building

The latency measured at Michigan DOT's Signal Shop at Lansing, Michigan, is (Figure 5.42):

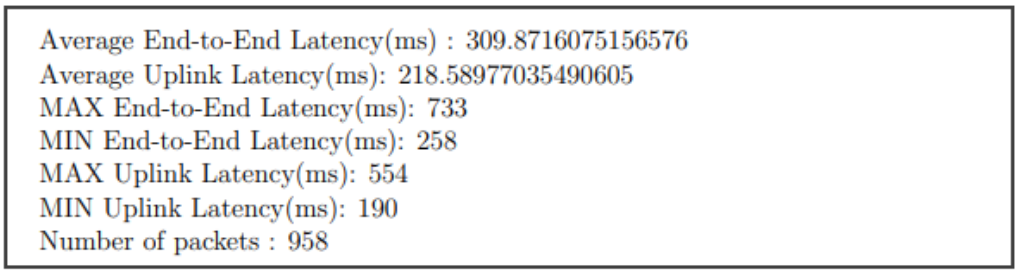


Figure 5.42: Latency measured using the app located at MDOT's Lansing Signal Shop, Michigan

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

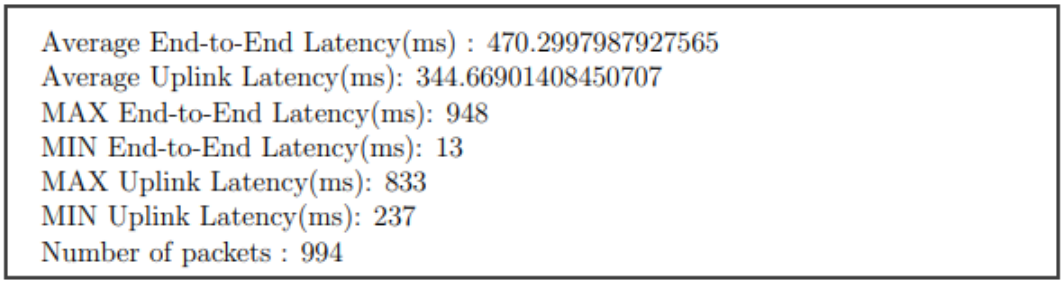


Figure 5.43: Latency measured using the app located at King St./M52 intersection, Owosso, MI

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 varies depending on network conditions, processing times, and other factors. While aiming for minimal latency is ideal, it is 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 backend workflow of the designed app is shown in Figure 5.44. The front-end view of the app is shown in Figure 5.45.

5.6 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 current project’s requirements. In this regard, ”receive data.py” in the git repository of section 3 realizes 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 messages.

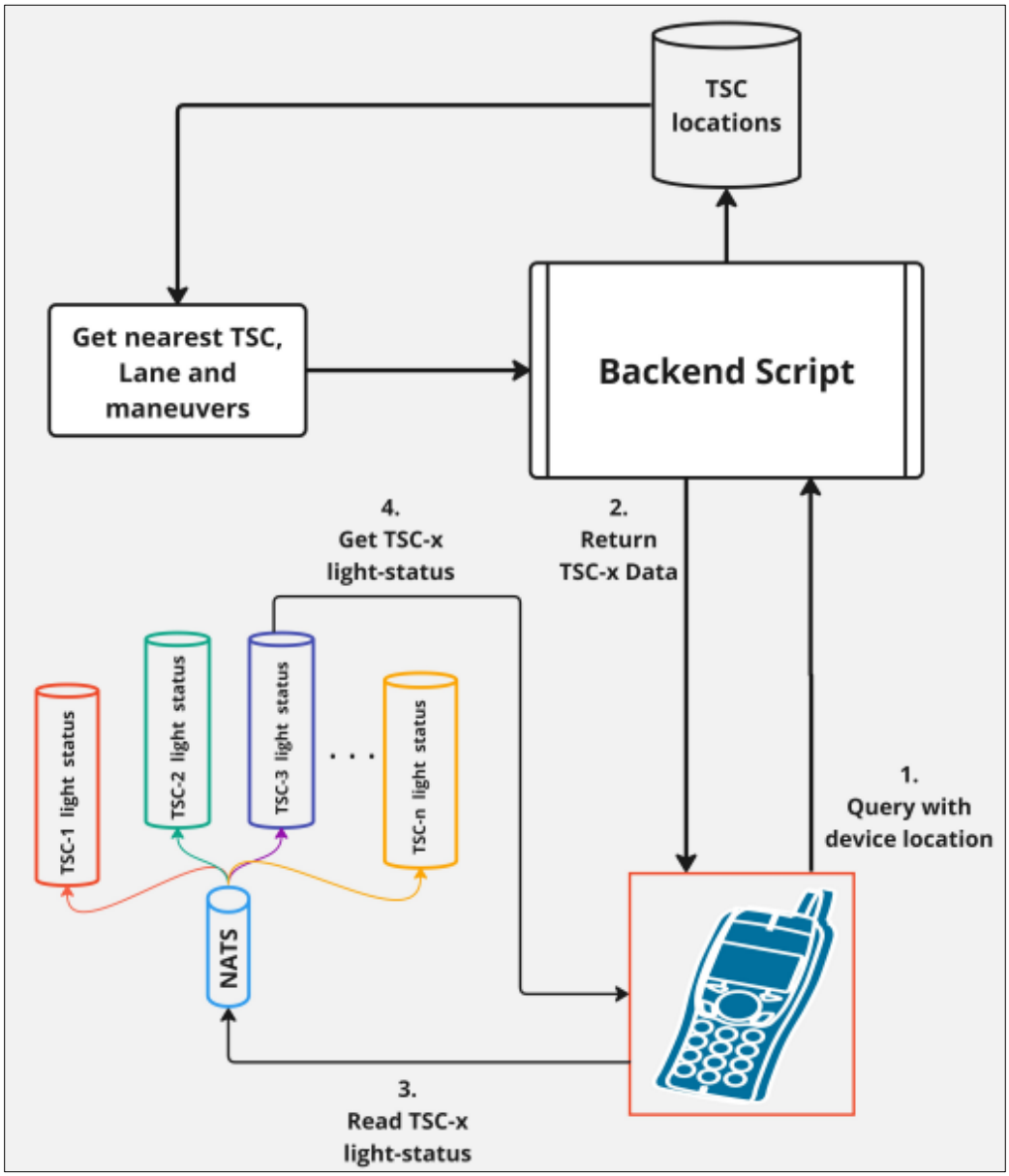


Figure 5.44: Web-App Back End Workflow

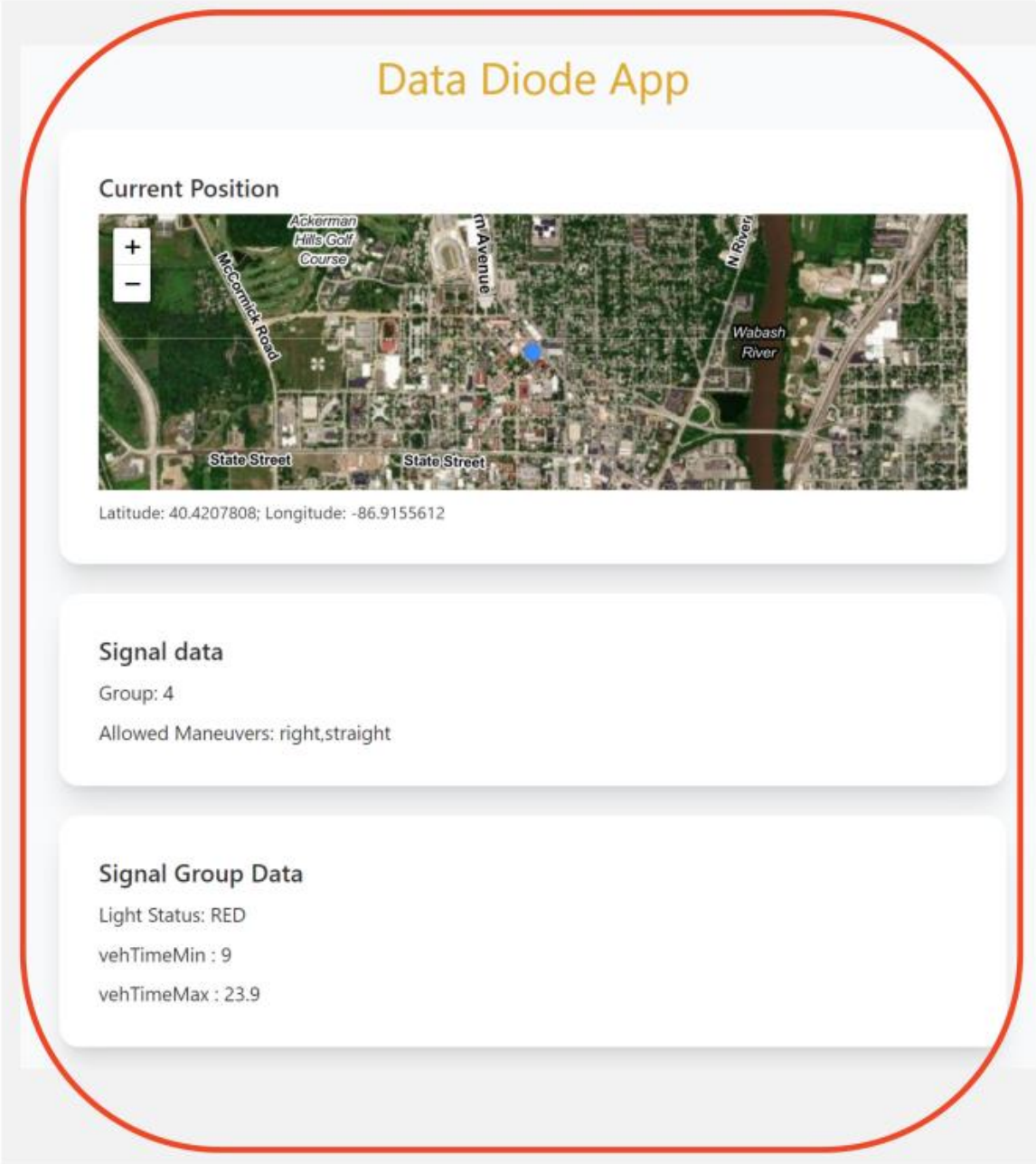


Figure 5.45: Web-App Front End View

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. Realizing this on AWS is explained in detail in Section 5.7. 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 5.46.

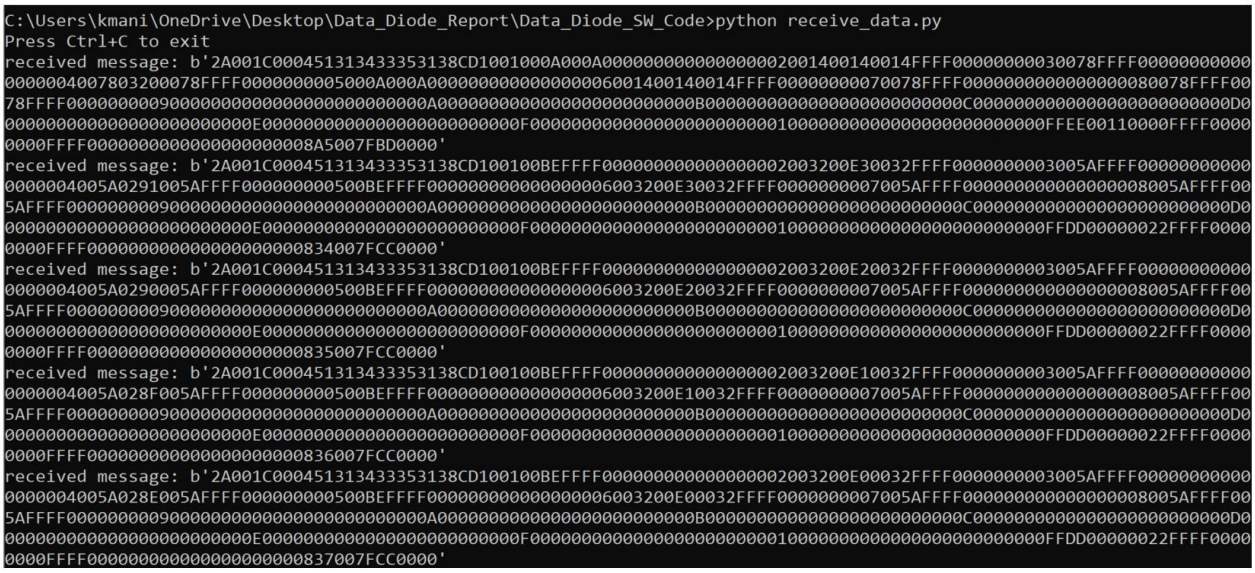


Figure 5.46: Python based Server's Console Output

5.6.1 Data Logging on to a Text File

The same "receive data.py" described in Figure 5.46 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. Where there exists adequate number of USB ports, all the three data loggings, i.e., the cell modem text file logging and the two STM32 serial interface debug loggings can be realized on the same PC.

5.7 Running the server on Amazon EC2 Instance

As briefly discussed earlier, Amazon EC2 Instance provides a public IP which could 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 realize the above Public-Pvt interconnect on the 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 the IP addresses allowed.
5. In the EC2 linux terminal run "receive data.py" described in 5.2.4 with port number and interface address same as that established 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 are shown in Figure 5.47:

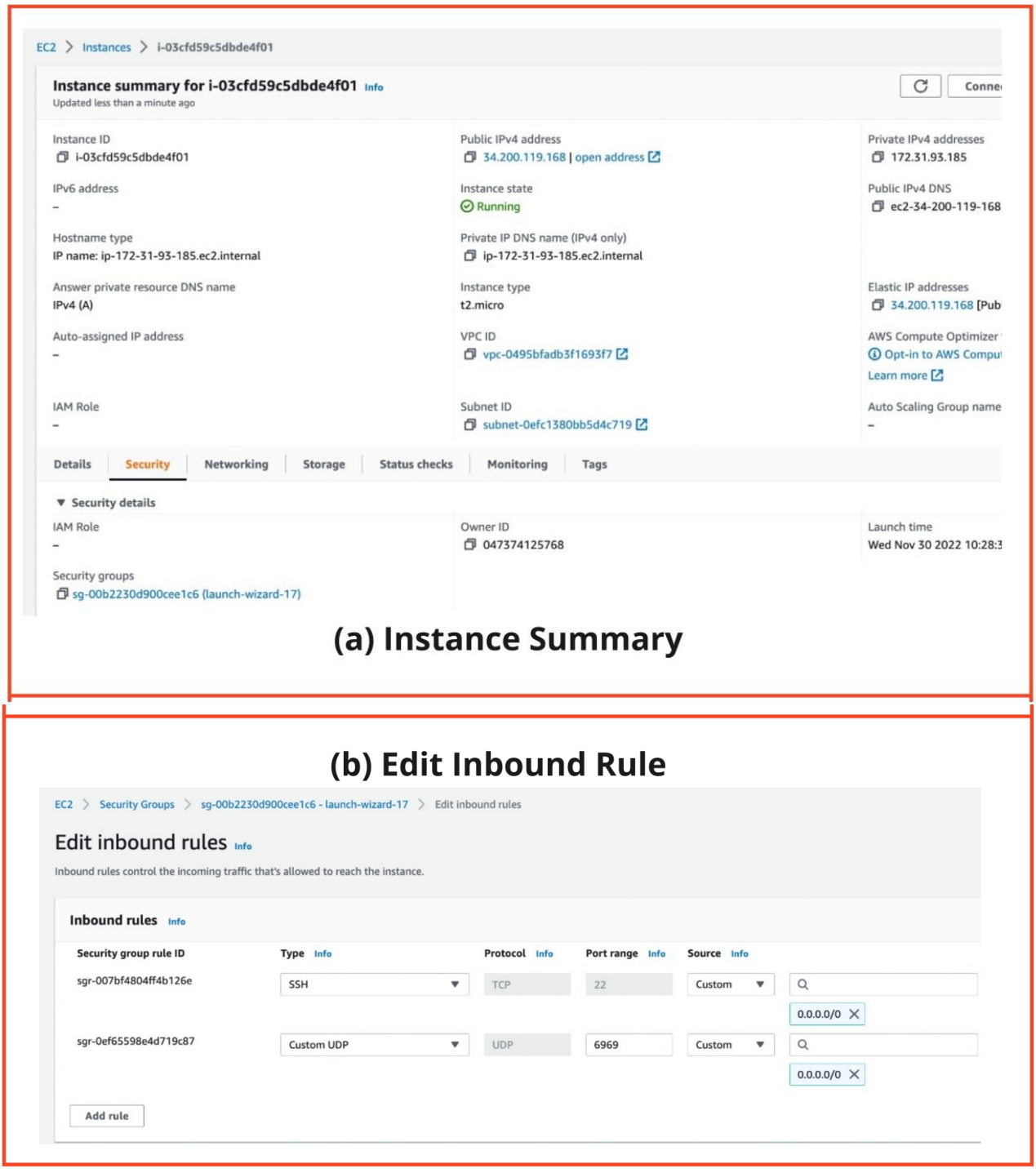


Figure 5.47: Creating Inbound Rule on an Amazon EC2 Instance

5.8 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, the team used 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 5.48 and Figure 5.49. The SQL files associated with the dashboard can be found in Grafana folder in the Githib repo.

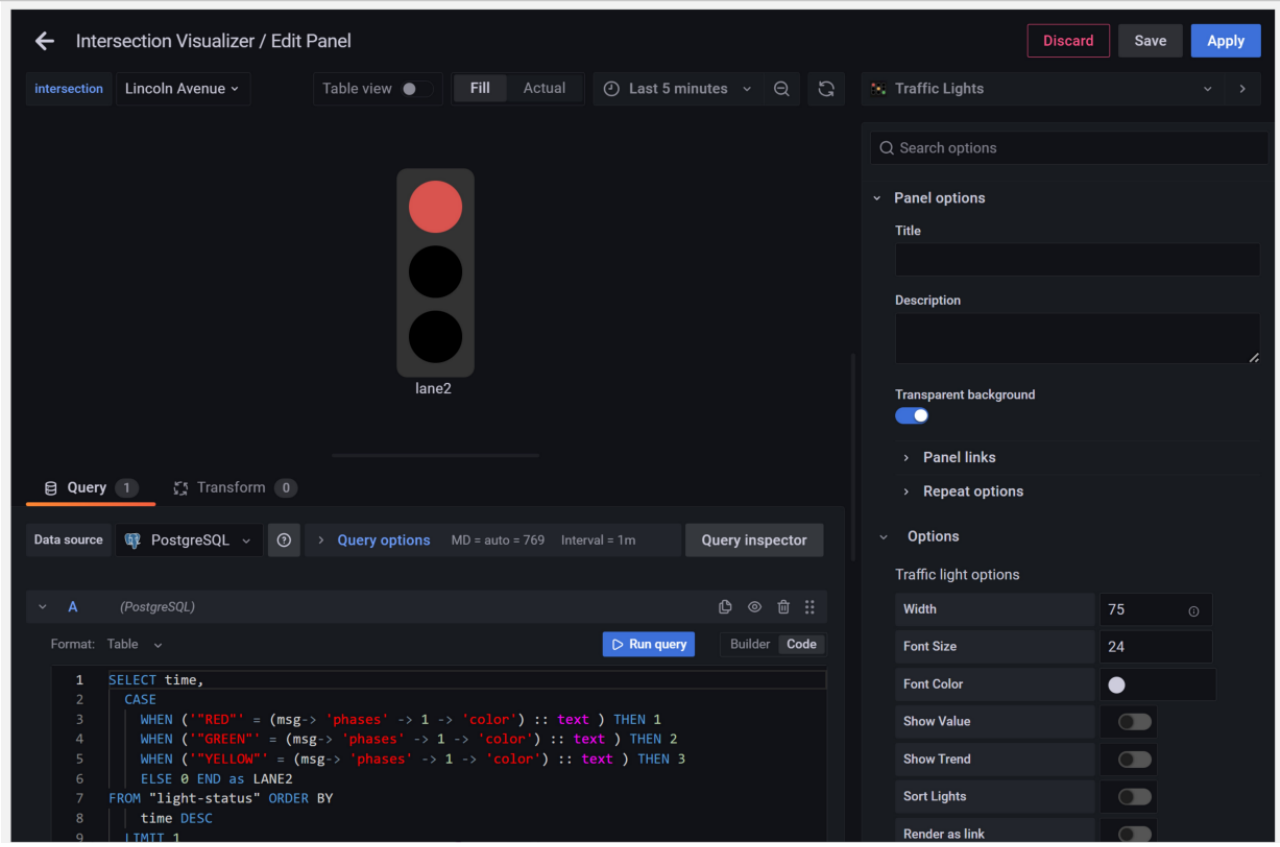


Figure 5.48: Grafana Traffic Light Plugin

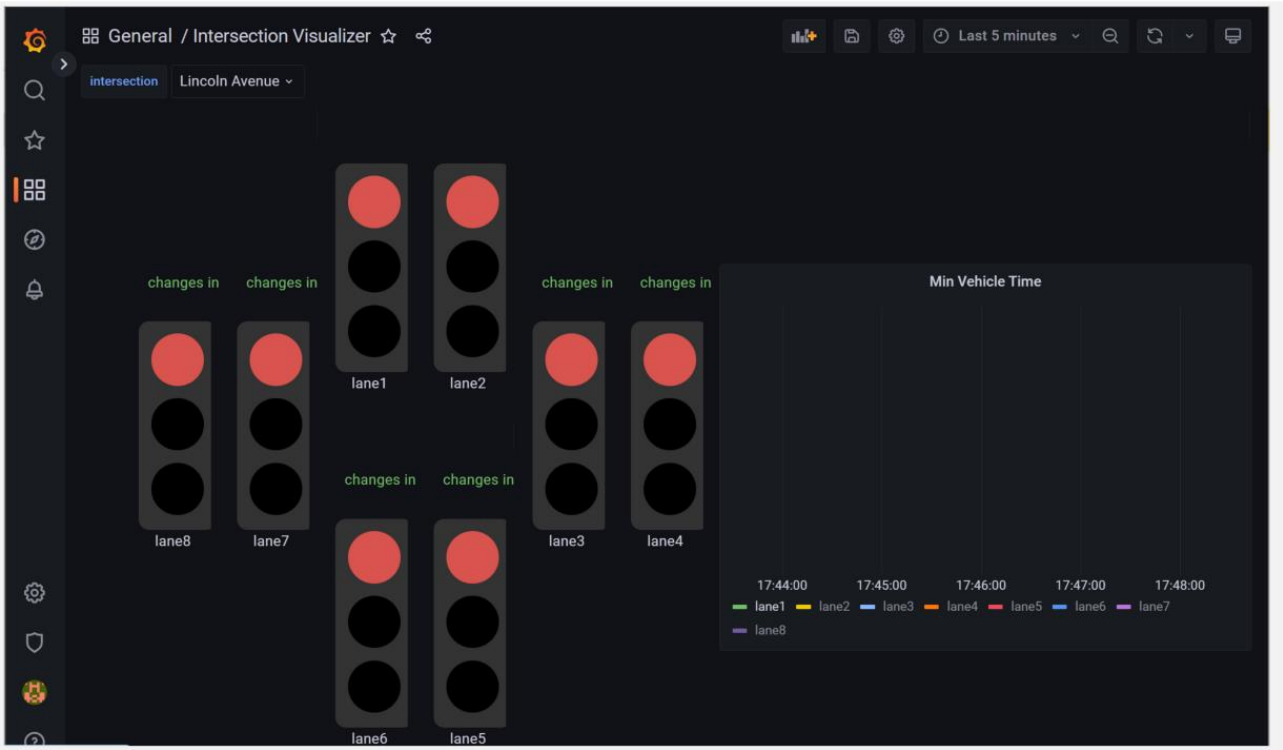


Figure 5.49: Grafana Dashboard

5.9 Diode Component Enclosures

Two enclosures were 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 5.50.

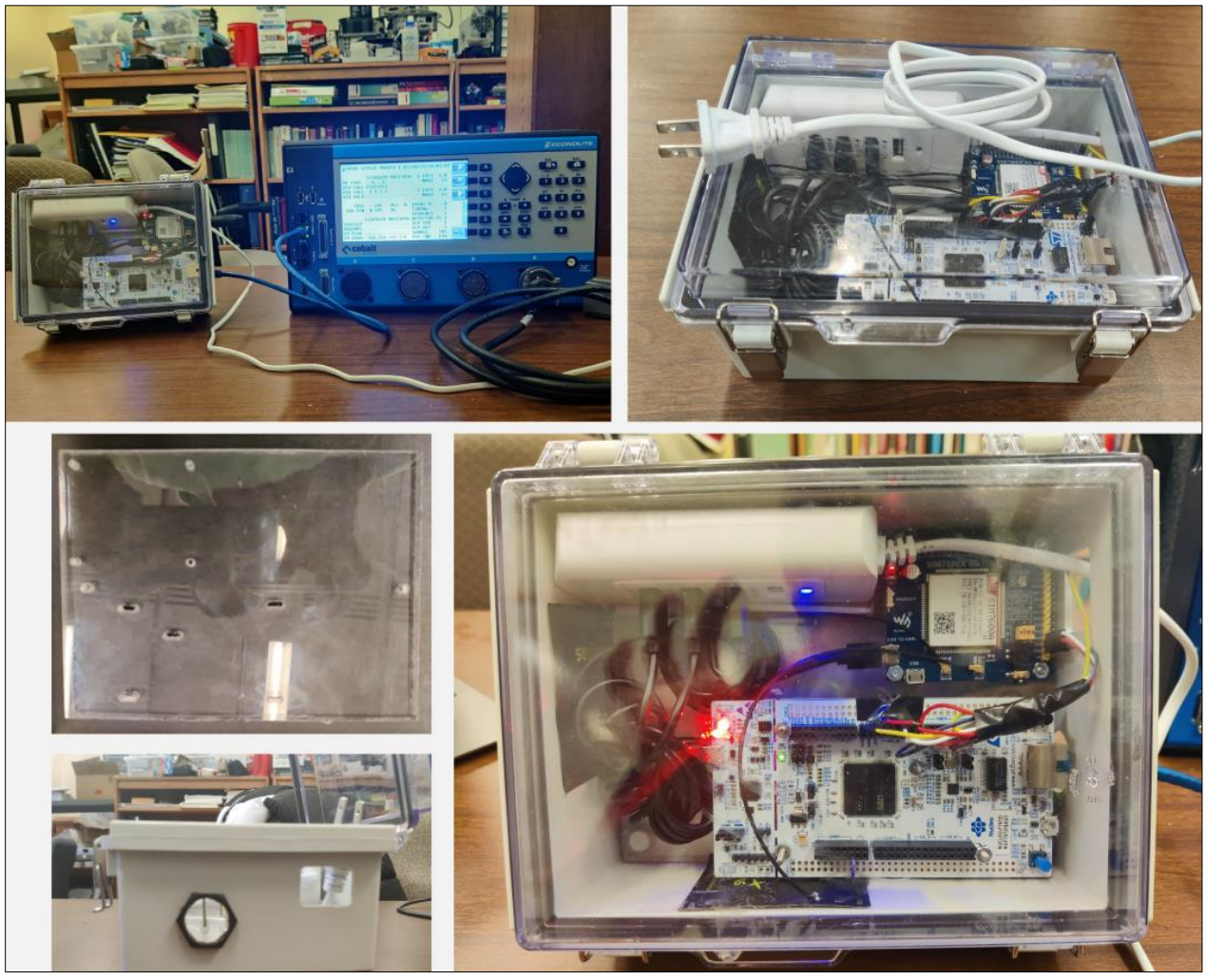


Figure 5.50: Diode Enclosure Version 1.0

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 Professor Montasir Abbas and David Hong of Virginia Tech. The CAD files for this enclosure design are also available in the GitHub repository under the folder Diode Enclosure Version 2.0. Notably, the enclosure 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 connected to the PCB using wires. This is a feature of STM32 Nucleo boards.

A notable feature of this version are the jumper pins for the power supply arrangement. The JP3 jumper on both STM32 boards needed to be in the E5V position. It is 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. The UART selection jumper on the cell modem must be in position A; this is meant to ”access raspberry pi via USB to UART”.

CHAPTER 6. CONCLUDING REMARKS

6.1 Summary

Cost-effective ways to obtain and distribute traffic data, and thereby, to facilitate operations decisions. This includes a need to make Signal Phase and Timing (SPaT) and MAP data more useful to mobile and other devices and connectivity infrastructure. Unfortunately, existing data collection and delivery methods are time consuming and costly. Therefore, to facilitate traffic operations in the current era of human-driven vehicles (HDVs) and the prospective era of connected autonomous vehicles (CAVs), this research developed a device to acquire, process, and disseminate SPaT data from Traffic Signal Controller (TSC) cabinets in a manner that ensures system integrity and yet fosters accessibility of timing information for traffic data end-users.

The device, referred to as a "Data Diode," consists of two interconnected microcontrollers operating through simplex communication (one-directional data flow). The device connects the TSC cabinet and the end-users device (such as a user-held device (UHD), thus facilitating the extraction and processing of SPaT data. The process begins with one microcontroller establishing a connection with the TSC's Ethernet port. Then, utilizing the Simple Network Management Protocol (SNMP) interface, the device securely acquires the SPaT data which then undergoes Cyclic Redundancy Check (CRC) encoding to ensure integrity protection and subsequently transmitted through a unidirectional UART interface to the second microcontroller. No SNMP commands can be passed back to the signal controller from the Internet. The interface cannot be used as an attack surface.

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. A battery of such TSC-Diode systems collect such data from various intersections, and the collected data are routed through their respective channels within the NATS system, thereby facilitating efficient organization and streamlined access.

A backend script acquires GPS coordinates from the UHD, 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 UHD. The app dynamically presents real-time information about the traffic signal status and precise timing of signal phase transitions for the lane of interest.

Overall, the integration of this technology into human-driven or connected autonomous vehicle driver assistance systems can help smoothen arterial traffic flow and transform urban mobility.

6.2 Discussion on Practical Usefulness and Future Work

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.

CHAPTER 7 SYNOPSIS OF PERFORMANCE INDICATORS

7.1 USDOT Performance Indicators Part I

Three (3) transportation-related courses (two (2) at Purdue and one (1) at Virginia Tech) were offered annually during the study period that was taught by the PI and a teaching assistant who are associated with the research project. Two (2) graduate students participated in the research project during the study period, two (2) Purdue students and one (1) Virginia Tech students. Three (3) transportation-related advanced degree programs (one (1) MS program (Purdue) and two (2) doctoral programs (one each at Purdue and Virginia Tech) utilized the CCAT grant funds from this research project, during the study period to support the graduate students.

7.2 USDOT Performance Indicators Part II

Research Performance Indicators:1 conference presentation was produced from this project. The research from this advanced research project was disseminated to 23 people from industry, government, and academia, through the conference presentation.

Leadership Development Performance Indicators

This research project generated 3 academic engagements and five (5) industry engagements (The City of Owosso, The Transformation Network, Michigan DOT, Cubicon Inc., T-Mobile). The PIs held positions in 2 national organizations that address issues related to this research project.

Education and Workforce Development Performance Indicators

The methods, data and/or results from this study were incorporated in the class content of:

- **Virginia Tech:** The Fall 2022 version of the following courses in the undergraduate civil Engineering program: CEE 5694: Traffic Signal System Operations & Control
- **Purdue University:** The Fall 2022, Spring 2023, and Fall 2023 versions of the following courses in the undergraduate civil engineering program: (a) CE 299 (Smart Mobility), an optional undergraduate-level course, and (b) CE 398 (Introduction to Civil Engineering Systems), a mandatory undergraduate course.

The students in these classes will soon be entering the workforce. Thereby, the research helped enlarge the pool of people trained to develop knowledge and utilize the technologies developed in this research, and prospectively, to put them to use when they enter the workforce.

Collaboration Performance Indicators

During this project, there was collaboration between the main institution (Purdue University) and other partners, as follows:

Five (5) industrial partners: The Michigan Department of Transportation, The City of Owosso, The Transformation Network, Cubicon Inc., and T-Mobile.

One (1) government agency (USDOT Volpe Center)

One (1) tertiary institution (Virginia DOT)

Two (2) academic institutions that provided matching funds.

The outputs, outcomes, and impacts are described in Chapter 8.

CHAPTER 8 STUDY OUTCOMES AND OUTPUTS

8.1 Outputs

Good and reliable data are central to any traffic management or operations effort. It has been demonstrated, through a field deployment in the city of Owosso, that the main study output (a novel data diode device), can contribute towards the efficiency and effectiveness of urban road transportation systems operations and management. The study processes and findings have been presented in a conference. Elements of the study will continue to be included in related graduate and undergraduate transportation engineering courses at Virginia Tech and Purdue University.

8.1.1 Publications, conference papers, or presentations

Conference Presentations
Gowda, M., Fehr, W., Balmos, A., Ajagu, R., Abbas, M., Krogmeier, J., Labi, S. (2023). Secure Acquisition of Intersection Data for Connected Vehicle Operations, The Third Annual Next-Generation Transport Systems (NGTS–3) Conference May 16–18, 2023. West Lafayette, IN.

8.1.2. Other outputs

(a)At Purdue, the research product was used to:

- help teach relevant concepts in two (2) Purdue undergraduate-level courses: CE 299 (Smart Mobility) and CE 398 (Introduction to Civil Engineering Systems).
- support future research related to the subjects of UAV-CAV network, trajectory planning and operational monitoring for CAVs.

(b) Virginia Tech developed methods for vehicle-infrastructure integration in an educational setting. The components of this initiative included this CCAT project on the data diode development. The educational initiative included conceptual design of an intersection control system for connected vehicles, and the creation of illustrative concepts for a driver warning system in a microsimulation environment, designed for an interactive classroom setting.

(c) The synergy in the collaboration between Virginia Tech and Purdue's UTC research facilitated innovation in connected automated vehicle technology and development in educational systems. Virginia Tech assisted Purdue's data diode project team in the creation of MAP (intersection layout and geometry) units, which provided reference points for the tracking of the vehicle trajectory as connected vehicles move along a corridor.

(d) Virginia Tech's team also provided packaging solutions using 3D-designed (using SolidWorks) and printed hardware housing for the connected vehicle data diode system. This involvement also allowed Virginia Tech's team to help create one of the cutting-edge connected vehicle V2I communication systems and inspired the creation of a proposed small-scale intersection control system for educational purposes.

(e) The conceptual design of an educational intersection control system involved the use of Electronic Design Automation (EDA) software for designing the circuitry consisting of microcontrollers and microprocessors. This stage of the project is designed to discover the methods for showing the working mechanisms of a V2I system with a low budget for the classroom setup.

(f) At Virginia Tech, the research process and product were incorporated into coursework, as follows:

- Courses taught related to the project or where the project material was used in part of the course: CEE 5694: Traffic Signal System Operations & Control
- Three (3) short courses developed and delivered by CCAT PI's (Montasir Abbas and Walt Fehr) to audience sizes ranging from 7-10 students:
    - Cubicon Design Methodology Part 1
    - Cubicon Design Methodology Part 2
    - Cubicon Design Methodology Part 3

(g) Collaborations

U.S. DOT Volpe Center: Walt Fehr is the creator of the data diode concept. Walt and other Volpe Center staff's key role included that of Interactive Evaluator. Interactive Evaluation is a technology transfer process being developed by the U.S DOT's Volpe Center to investigate promising new technology and bring it into the transportation sector if it meets its claims. The process starts with the evaluator engaging with the technology developer as a subject matter expert to fully understand the technology and to develop specific metrics that will be used to confirm or refute the developer's claims – better, faster, cheaper, etc. When the developer then proceeds to a demonstration designed to showcase the technology's claims, the evaluator will assume the more traditional role of an evaluator to objectively measure the claims. The transportation community will then have a better understanding of the new technology and a measure of the potential benefit it might bring.

Michigan Department of Transportation (Lansing Signal Lab): The Michigan Department of Transportation (Lansing Signal Lab) bench agreed to test the developed device. Plans are underway to test the device (in Phase II of this CCAT project) at the MDOT Traffic Signal Lab in Lansing, and to field test it in Owosso, Michigan. The MDOT engineers involved in this effort and these plans include Mr. Nathan Bouvy, Scott Holzhei, Travis Phillips, Hilary Owen, Ryan Schian, Ross Venable, and James Kwapiszewski.

T-Mobile: Providing transport media for transportation infrastructure data are increasingly becoming an important service for the T-Mobile organization as the company strives to support connected intelligent transportation as part of the Internet of Things. The company expects to acquire wider knowledge of the needs of the transportation community by participating in this project. In turn, T-Mobile's expert suggestions and access to our data transport medium help strengthen the project goals. T-Mobile offered their extensive up-to-date wide area network service, for incorporation in the research. This will be particularly valuable in the follow-up testing and deployment phase where a focused performance evaluation of the developed device, in the context of its prospective role in infrastructure data distribution, will be carried out.

Transformation Network: David Acton is the President of The Transformation Network, Inc. (TTN). The Transformation Network's role was to provide technology and commercial development resources (including project management) to the deployment in Owosso, Michigan. Dave Acton is an internationally recognized subject matter expert in vehicle connectivity, Internet of Things (IoT) architecture design, and complex project deployment leadership.

Mayor Christopher Eveleth, City of Owosso, County of Shiawassee, MI: Owosso is a 15,000 person, four square mile, "micropolitan" city. The role of the City of Owosso is to be the project deployment location. Throughout its history, Owosso has been home to many inventors, industrialists, authors, politicians, and technology pioneers. A common thread in Owosso is that residents through the decades have acknowledged that geography is not the critical factor in determining success of various public initiatives. Rather, success is dependent on a community's ability to collaborate. Since 1836, Owosso has demonstrated that the city is able to align all necessary stakeholders quickly and decisively to achieve success. This project has further strengthened that tradition. In that spirit, Mayor Christopher Eveleth has agreed to provide access to the City of Owosso's traffic infrastructure in summer 2023 for testing the device.

Sandy Klausner (CubeFog Corporation): Sandy Klausner is the principal developer of the Cubicon Technology Suite. This phase of the project was unable to investigate two specific claims about Sandy Klausner's Cubicon technology. (1) that the Systems Engineering Language will help transportation practitioners better describe the desired system behaviors so that more efficient and economical implementations will result, and (2) that Cubicon technology will provide highly-efficient data distribution techniques that will synchronize the deterministic components of a distributed system using already-existing communication media. It is expected that this will be done in the next phase of the project (full deployment). The next phase will be conducted in part of the larger ad-hoc group's demonstration installation centered on the distribution of traffic signal state data and other infrastructure data for an entire community using already-existing wide area network communication. The installation and operation of the demonstration installation is expected to be significantly cheaper and faster to install and lower total cost-to-operate than similar installations that make use of short-range wireless communication without any sacrificed reduction in data integrity or delivery performance.

8.2 Outcomes

The success of this project could be attributed to the diversity of the research team. The team consisted of representatives from three key stakeholders of any transportation initiative: government (Michigan DOT, City of Owosso), industry (T-Mobile, The Transformation Network, Inc.) and academic (Purdue University, Virginia Tech). These participants worked together to design the study and to plan the prospective (Summer 2023) pilot deployment of the developed product. The expected outcomes of this project are the prospective changes that the economic acquisition of intersection data could bring to the road transportation system, or its regulatory, legislative, or policy framework. These are:

- City authorities and road public agencies are now provided the data diode device, a novel and cost-efficient way to drastically reduce the cost (while maximizing the effectiveness) of their data collection tasks at their intersections.
- Considering the effect of scale economies, expanded deployment will serve to further reduce the production costs of the data diode device.
- The research outcome is consistent with USDOT's strategic goals of innovation, cost-effectiveness, safety, and mobility. The device is novel in its functions. The project's explicit consideration of cost is rather unique because it addresses an often-overlooked aspect of ITS deployments – cost.

8.3 Impacts

A list of specific impacts from this research project, are as follows:

- The low costs of production, installation, and operations of the developed device will foster advancement of the connected intelligent transportation ecosystem when the potential benefits of the device become more widely recognized.
- The two Purdue graduate students and the Virginia Tech graduate student who worked on (and were funded by) this project will enter the workforce in 2024-2025 and are expected to help implement and/or improve the data diode device developed in this project.
- Students that took Virginia Tech's course CEE 5694 (Traffic Signal System Operations & Control) and those that participated in the 3 Cubicon workshops, will enter the workforce in 2024-2025 and are expected to help implement and/or improve the data diode device developed in this project.

REFERENCES

[1] S. Ritchie, C. Rindt, and D. Deeter, 'Technological Innovation and Intelligent Transportation Systems for the US: Perspectives for the 21st Century', in US Infrastructure, Routledge, 2019, pp. 37–54

[2] 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

[3] C. Fernando, Designing Microservices Platforms with NATS: A modern approach to designing and implementing scalable microservices platforms with NATS messaging. Packt Publishing Ltd, 2021

[4] W. Gay, 'Beginning STM32', Beginning STM32, 2018

[5] A. Kommu and R. R. Kanchi, 'Designing a learning platform for the implementation of serial standards using arm microcontroller LPC2148', in International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), 2014, pp. 1–6.

[6] T. Urbanik, A. Tanaka, B. Lozner, E. Lindstrom, K. Lee, S. Quayle, S. Beaird, S. Tsoi, P. Ryus, D. Gettman, S. Sunkari, K. Balke, D. Bullock, CHRP Report 812: Signal Timing Manual 2ND Ed., Transportation Research Board of the National Academies, Washington, D.C., 2015.

APPENDIX

APPENDIX 1: List of resources and weblinks

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