Optical Wireless Communication

3rd IEEE Susquehanna Section Capstone Award – 2023

Table of Contents

I. Abstract

In this capstone project, we investigate a mobile optical wireless networking system enabling reliable transfer of data among multiple nodes without any prior knowledge of each other's location. Each node is equipped with a multi-transceiver optical wireless communication (OWC) module with electronic beam steering capability. The multi-transceiver design facilitates in establishing multiple simultaneous OWC links with different neighbor nodes. The proposed system ensures error-free data transfer through fast line-of-sight (LOS) discovery, robust maintenance of OWC links, and utilizing reliable communication protocols. We also present a preliminary prototype of our multi-transceiver multi-node OWC system. We evaluate the effectiveness of the proposed system by analyzing the results obtained from test-bed experiments using the developed prototype. The results demonstrate that the proposed system can enable reliable communication among multiple mobile nodes.

II. Introduction

Optical wireless communication (OWC) systems have the potential to complement traditional omnidirectional radio-frequency (RF)-based wireless technologies and help in addressing the RF spectrum capacity crunch. OWC has attracted significant interest in recent years due to its use of the unlicensed optical spectrum and ability to provide much higher wireless communication speeds compared to RF. The high directionality of OWC transceivers enhances signal security and lowers the probability of interception or detection. It can also help establish communication RF-unfavorable environments, e.g., presence of jamming. OWC transceivers also improve spatial reuse and enable parallel communication links. These advantages of directional transceivers makes OWC suitable for tactical ad hoc networks with multiple nodes simultaneously transmitting high bandwidth data streams.

The advantages of OWC can be beneficial for networks with mobile nodes deployed for either civil or military missions. Rapid communication deployment in disaster recovery, such as monitoring forest

fires, sharing of high resolution imagery for precision agriculture and next generation air traffic control, and secure command and control of mobile units in combat are a few examples of such missions. All of these scenarios may generate a large amount of data which can be transferred at very high speeds using OWC transceivers. Thus, mobile unmanned ground and aerial vehicles equipped with OWC systems are emerging as a promising technology.

Despite the advantages, mobility of nodes can result in transient OWC link loss. An OWC transmitter and receiver must be aligned to establish line-of-sight (LOS) between each other. OWC also requires this LOS to be strictly maintained for preventing the link loss. Hence, LOS discovery and maintenance are crucial for mobile OWC networks.

III. Project Specifications

Hardware

General Functionality

The hardware on this project will provide the physical means for wireless optical communication. The communication will take place between individual systems, which will be referred to as robots in this paper. Each robot will run software to implement the algorithms, and the software will be used to control infrared transceivers. A printed circuit board will be created so that the assembly of each robot is consistent.

Communication Process

General Communication

Fundamentally, all of the processes in this project will be accomplished through the use of optical wireless communication. It is with this communication that the discovery process will be done, and this process will be the foundation on which the transmission of the payload and the maintenance of the connection is conducted. The process involves using software to convert digital data into signals that

can be understood by the hardware, and the hardware will convert the signals into infrared light. The light will be transmitted by using air as the physical medium. The communication process is not specific to an individual part of the system, and all parts of the system have the ability to act as transmitters and receivers.

Establishing and Maintaining Connections

The individual systems (robots) in this project should be able to automatically connect with one another, and once a connection has been established they should be able to maintain it indefinitely. The communication network has to be able to support any number of robots and the network should only be limited by the amount of hardware in the system. Nothing in the communication process should be statically implemented, and each robot should have the ability to dynamically act as a server and a client.

Payload Transmission

Once a connection has been established between two robots, a payload will be transmitted between them through the use of optical wireless communication. The payload should be able to be constructed from any file type, and there should be no restrictions on what kind of data can be sent across the network. The payload will be sent through a reliable connection, and any corrupted, lost, or malformed data will be automatically retransmitted. That is, the payload will be perfectly reconstructed on the receiving end of the connection, and it will be a perfect representation of the original payload.

Discovery and Maintenance

Discovery

A robot may discover another robot through initiating or receiving a connection. Robots establish connections with each other using a TCP handshake. For instance, a client robot initiates the TCP connection by transmitting a SYN segment, indicating its intention to establish a connection with a server robot. The server robot, upon receiving the SYN segment, responds with a SYN-ACK segment. The SYN-ACK segment acknowledges the client robot's request and signals the server robot's readiness

to establish the connection. Finally, the client robot acknowledges the SYN-ACK segment by sending an ACK segment. This completes the TCP handshake process, and the connection is established between the client and server robots. Each robot fills the role of both a client and a server, and therefore simultaneously listens for incoming connections while also initiating connection requests. Once a reliable connection is created, the TCP socket is used for both sending and receiving payloads between the robots.

Maintenance

The connection between robots is monitored through the utilization of the ICMP network protocol. By leveraging ICMP's echo request and echo reply messages, commonly referred to as "ping," the system assesses the reachability of the other robot. For instance, the initiating robot can send an ICMP echo request to the counterpart robot. Upon receiving the echo request, the counterpart generates an ICMP echo reply and sends it back to the initiating robot. By confirming the receipt of an echo reply for each transmitted echo request, the initiating robot can evaluate the counterpart's reachability.

To determine the reachability of the other robot through each of the transceivers specifically, the initiating robot gives each transceiver a unique echo request to transmit, such that the number of received echo replies corresponds to the specific transceiver's connection strength with the other robot. A transceiver's unique echo request that generates the most echo replies is selected as the best transceiver for payload communications with the other robot. Transceivers that are not able to communicate with the counterpart result in zero echo replies being generated for that transceiver's echo request. Consequently, the maintenance strategy ensures that a transceiver is only selected for payload communication if the transceiver's echo request successfully generates echo replies.

IV. Hardware Final Design Implementation

Hardware Components

TurtleBot:

The Kobuki TurtleBot from the previous group's implementation is still being used to carry and position the transceiver towers. There is no need to use anything different because the TurtleBot still is more than capable of meeting all of the requirements of this capstone project. The TurtleBot can take a vast number of different commands over serial communication, one of which allows for the program on the Raspberry Pi to have full control over the locomotion of the transceiver towers.

Transceivers:

The transceivers from the previous group are used in the final design because they are a perfect all in one package of things needed, it also eliminates the need to create transceivers. The previous orientation of the transceivers from the previous group was vertical with the pins pointing to the right which was determined to cut off a few degrees of the 48 degree coverage area, to fix that issue the orientation was changed to horizontal so that the pins point down to maximize the area of coverage. The transmitter in the Irda Click 3 transceivers uses an 850-900 nm wavelength to send data, which is the same wavelength range that the receiver on the transceiver can accept. The measured range of the transceivers is ~2m before the receiving robot starts to receive partially corrupted data and ~4.5m when it stops receiving data all-together. Used the transceiver window size data sheet to make sure our window size on the tower is big enough that we are not cutting off any of the area of coverage.

Serial Converters:

Each robot features 8 USB to TTL adapters all with unique serial numbers which gives more control over the robot. Each adapter serves the function of converting a USB connection to TTL, enabling the Raspberry Pi's to open a separate serial port for each of its transceivers. One of the major advantages of having a separate serial port for each transceiver is that if one of the serial ports has received data in its buffer, the program knows which transceiver specifically received the data. A separate serial port for each transceiver enables the creation of an algorithm that if a serial port has data, then that serial port can be used to communicate with another robot. Before any changes were made to the capstone project, the

previous group only implemented one serial port. Having one serial port is less optimal than 8 because the clarity of which transceiver received the data is entirely lost.

NeoPixel Ring:

In the design a twenty-four LED circular light array was added to the top of the robot. Each of the twenty four LEDs are individually addressable which allows the LEDs to light up any color using RGB values (0-255). Since there are twenty four LEDs and only eight transceivers, each transceiver has three transceivers at its disposal. Currently the middle of the three LEDs lights up to which transceiver the other robots are connected to.

Raspberry Pi 4:

In the previous design the two 4:1 multiplexers that were used to send and receive data from the transceivers were replaced in favor of a simpler design, in which it is connected into the Raspberry Pi. The eight transceivers are individually connected to a single serial converter through the PCB, the serial converters then plug into one of the two 4-port USB Hubs that plug into the Raspberry Pi. The Raspberry Pi then handles the discovery & maintenance methods and the sending & receiving of the data between two transceivers. The turtle bots came with laptops that were used to make the robots move randomly, that code was put into the Pi to reduce unnecessary items, now the laptops are no longer needed.

USB 3.0 Hub:

Each robot utilizes two 4-port USB hubs. The purpose of the USB hubs is to expand the number of USB ports that the Raspberry Pi's are able to support. Normally, a Raspberry Pi only supports 4 USB ports, which is not enough to have the 8 USB to TTL adapters plug directly into the Raspberry Pi. To solve this problem, 4-port USB hubs are used. Each USB hub is capable of hosting 4 USB-to-TTL adapters, and since there are 8 total transceivers, two USB hubs are required per robot. The reason 4-port USB hubs were chosen instead of an 8-port USB hub is because the 8-port USB hub required an

additional external power supply. After implementing the two USB hubs on the robot, the total number of USB connections that each Raspberry Pi can support is expanded to 10 total, leaving 2 USB connections left for anything else.

Power Bank:

The current power bank used to power the system but not the turtle robots, which use their own battery. The power bank is an Anker portable charger with a capacity of 25600 mAh, it has two USB type A 3.0 ports and one USB type C port. The USB type A 3.0 ports are able to charge at 18 W while the USB type C port can charge at 60 W. Currently the power bank's USB type C port is used to power the Raspberry Pi, which is where the transceivers, serial converters, USB hubs, and neopixel get their power.

High-Level Hardware Diagram:

As seen in Figure 1, the hardware components are connected in a cascading format. The Raspberry Pi is powered by the Anker Power Bank, connected via a USB-C to USB-C cable. The Raspberry Pi's USB 3.0 ports are directly connected to 2 of the USB hubs, effectively expanding the number of USB ports to a total of 10 ports (2 remaining USB ports on the Pi and 8 extended ports from the hubs). Connected to these 8 USB ports are the 8 USB-TTL adapters. The serial pins of these converters are inserted into the printed circuit board, which are then routed to the data pins on the 8 respective transceivers. Also routed from the Raspberry Pi are 4 voltage and data pins: 5 V, 3.3 V, GND, and GPIO 18. The 3.3 V pin is connected directly to the printed circuit board to supply power to the reset circuit. The remaining 3 pins are connected to the NeoPixel LED.

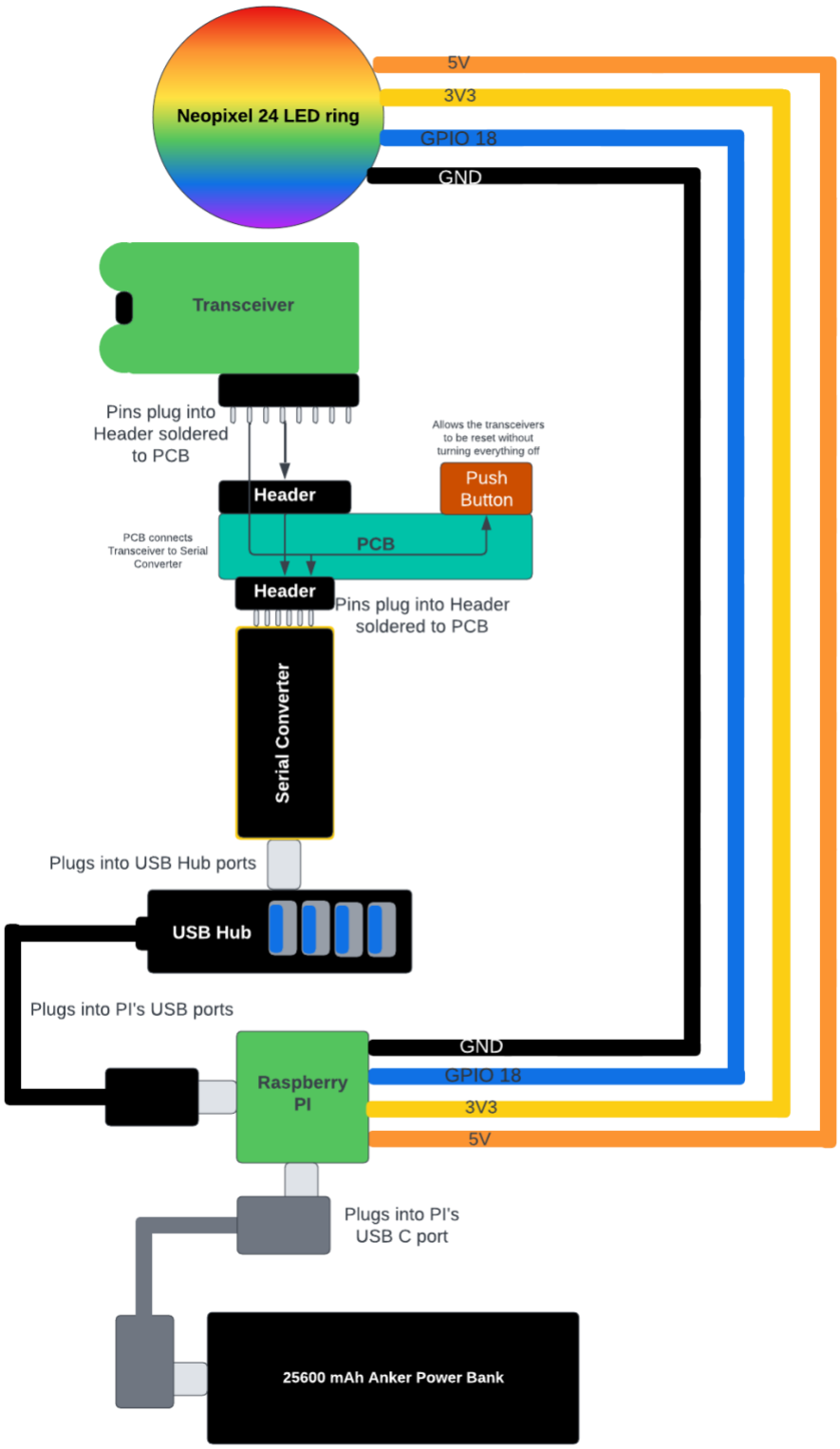

Figure 1: High-Level Hardware Block Diagram

Printed Circuit Board

A Printed Circuit Board (PCB) was designed in KiCad to directly connect hardware components without the use of excess jumper wires. The main purpose of the PCB was aimed to efficiently direct voltage and data pins from the transceivers to their respective serial converter. This direct connection was achieved by 0.25 mm wide traces that were placed on both sides of the 2-layer board. The paths of these traces were positioned and shaped to limit improper branching behavior, which in extreme cases, can lead to shorts in the circuit as well as possible heat traps at tightly-angled connections [5]. The traces are joined to through-hole pads, which are oriented to mimic the spacings and sizes of the devices' male connector pins.

The spacing between the through-hole pads was consistent between the transceiver and serial converter pins, and was determined to be 2.54 mm (0.1 in). Across the board, the sizes of the through-hole pads were chosen to utilize a universal 1.85 mm pad diameter with a through-hole diameter of 1.15 mm. Although the spacing dimensions are derived from the devices' datasheet, the through-hole sizes were chosen to loosely replicate the recommended layout dimensions of the female pin headers to which the hardware devices would attach. The recommended dimensions of the hole size was documented to be 1.02 mm [4], however, due to the team's inexperience with through-hole soldering, this measurement was increased to 1.15 mm. Likewise, the pad size was obtained from the standard 1.5 pad-to-hole ratio measurement of 1.725 mm based on the 1.15 mm hole size. This measurement was additionally increased for ease of soldering to 1.85 mm.

The arrangement of these through-holes was executed through customized device footprints designed in the KiCad software. Device footprints are meant to represent the physical layout of the PCB's components. Within the PCB Editor, the footprints were able to be aligned along a circular array, equally distancing the transceivers and accurately rotating them in increments of 45°. Likewise, a liner array was used to equally distance the through-holes for the serial converters within the converter hub footprint, as seen in Figure 2. A similar process of the through-hole layouts is used to generate the

footprint for the transceiver device, as seen in Figure 3. Because the serial converters were vertically inserted in a USB hub, the distance between the through-holes must have been equivalent to the distance between the USB sockets on the hub.

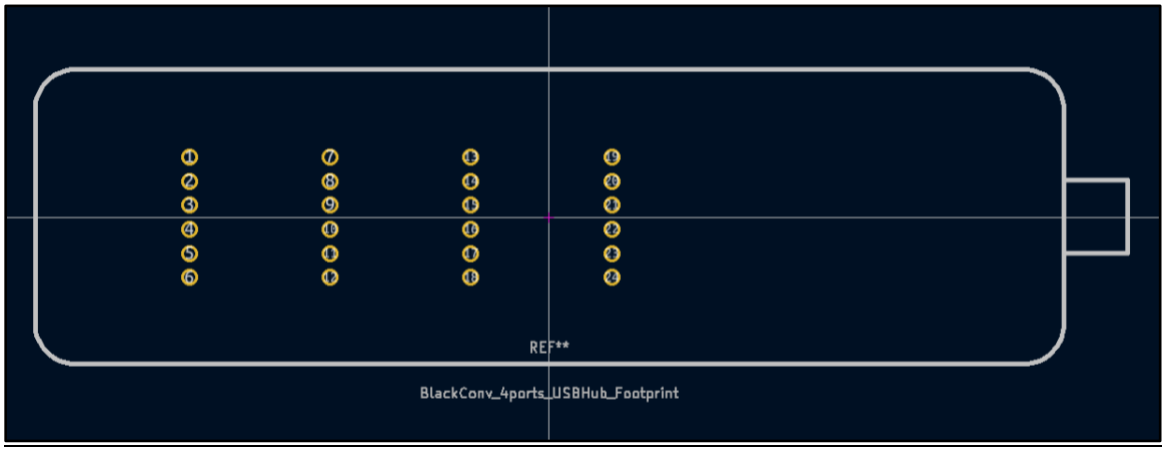

Figure 2: Converter Hub Device Footprint

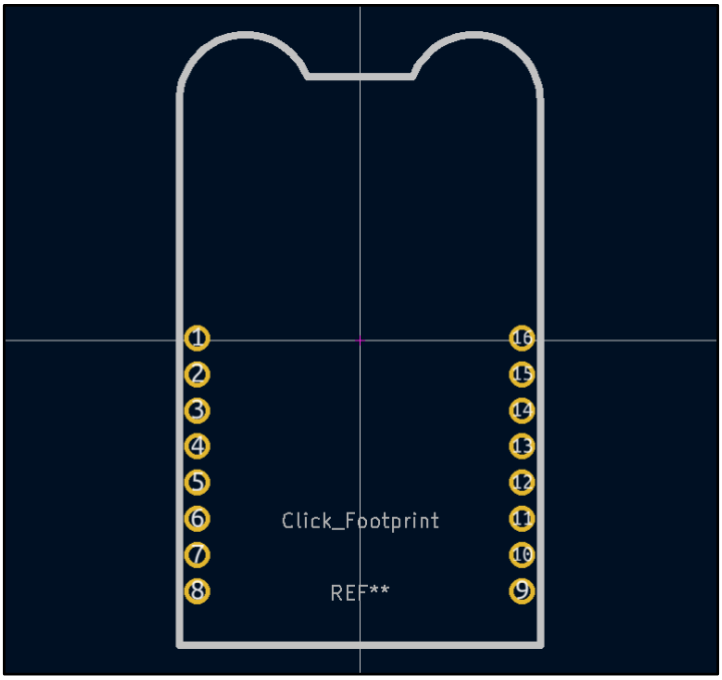

Figure 3: Transceiver Device Footprint

A footprint for the NeoPixel LED Ring was additionally designed, with accurate measurements sourced from the Ring's datasheet. Male 1x1 pin headers were soldered in the Ring's native through-holes in order for the LEDs to be inserted into the PCB's female header sockets. Although all 6 pins of the Ring were included in the footprint, only 4 through-hole pads were connected to board traces. This is due to the absence of any current use for the Ring's Data_Out pin, along with the second 5 V power pin. The NeoPixel LED Ring's footprint can be seen in Figure 4.

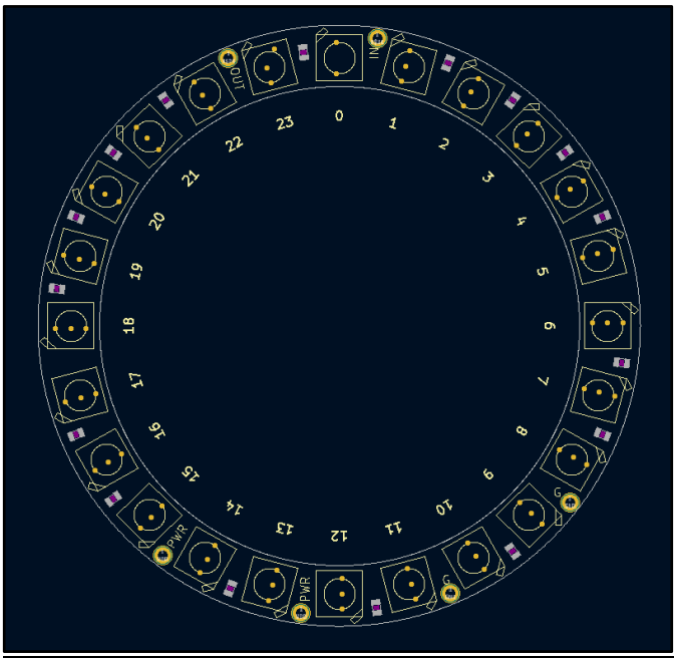

Figure 4: NeoPixel Ring (24 LED) Device Footprint

The traces throughout the PCB correspond to a dedicated wiring schematic that was additionally designed in KiCad. In order to organize the schematic in a clear and concise manner, customized symbols were created that represent the circuit pinouts of the PCB's components. It is within this schematic where the connections between specific component pins are designated. As seen in Figure 5 below, each transceiver is wired to its corresponding serial converter within either one of the two serial converter hubs. The VCC pin on the converter is directed to the 3v3 power pin on the transceiver, along with the converter's RX and TX pins wired to the transceiver's TX and RX pin respectively.

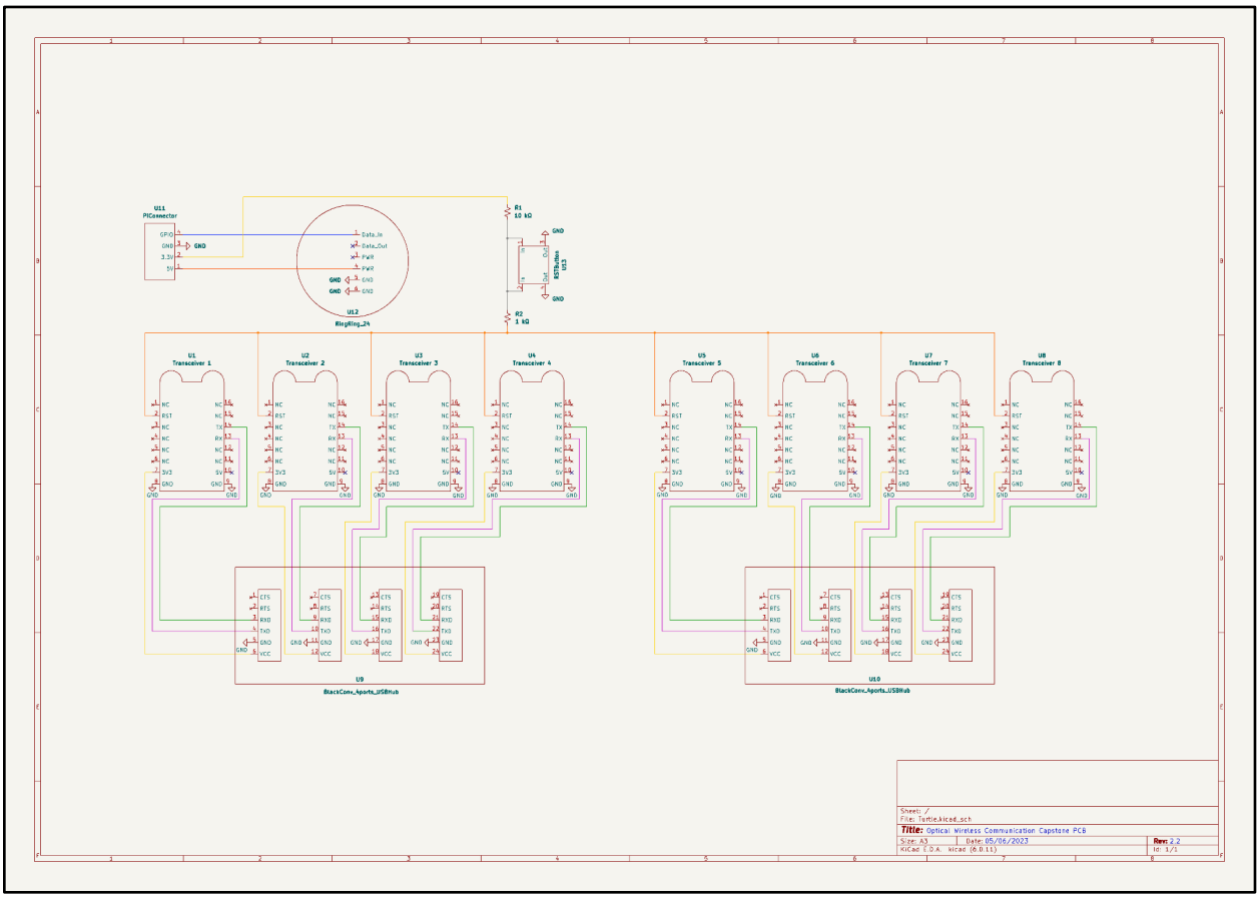

Figure 5: PCB Wiring Schematic

Additionally, 4 output pins from the Raspberry Pi are plugged into the PCB, referred to as the "PiConnector" symbol at the top of the circuit. Two of the Raspberry Pi's pins route directly to the NeoPixel: a 5 V power pin and a General Purpose Input/Output (GPIO) pin that controls the NeoPixel's LED logic. The third and fourth pins are the ground (GND) and 3v3 power pins. The GND pin from the Raspberry Pi is anchored to the GND planes on either side of the board. These conductive planes are where every device on the board is grounded, ensuring the entire circuit schematic utilizes a common GND. The Raspberry Pi's 3v3 power pin is routed to a tactile button after passing through a 10 kΩ pull-up resistor. When activated, the button is responsible for redirecting the Raspberry Pi's 3.3 volts to the GND plane rather than the reset (RST) pin on the eight transceivers. When the RST pin reads this low voltage state, the transceivers will enter its reset state. When the RST pin is brought back

to a high voltage state (3.3 volts), the device begins normal operation. This circuit allows the ability to manually reset the transceivers without requiring the entire board to be disconnected.

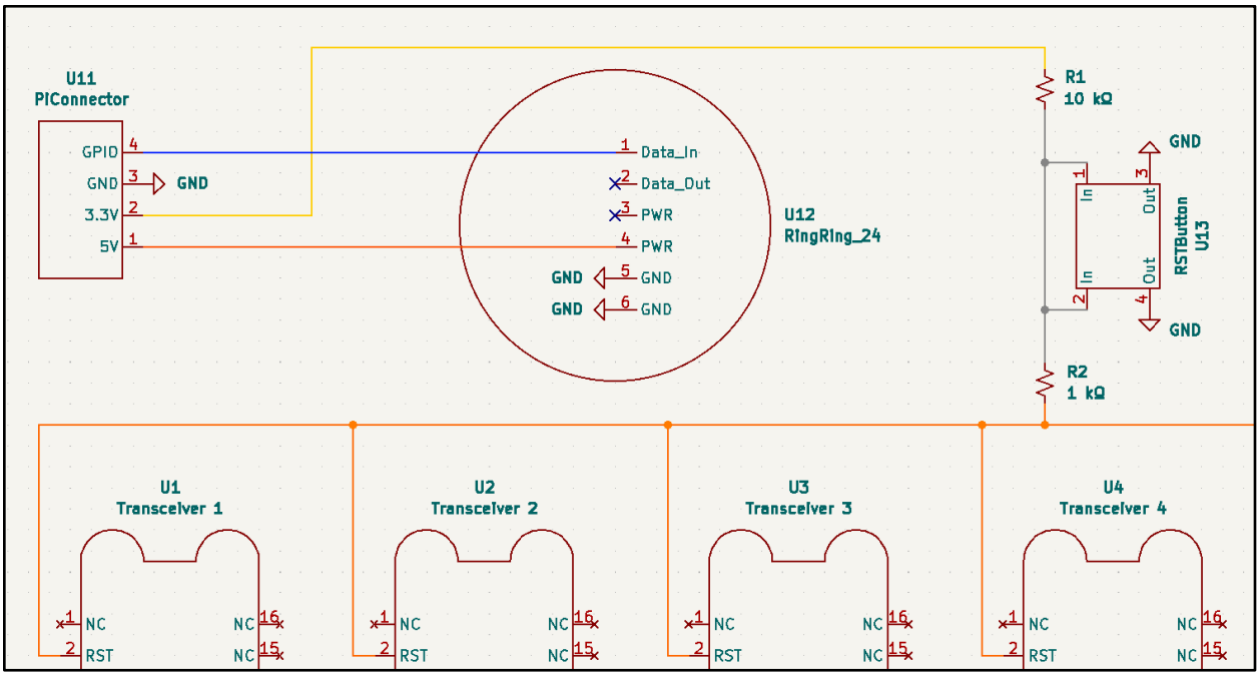

Figure 6: Raspberry Pi Connection with Reset Circuit

Beyond the implementation of device footprints and traces, the PCB editor was also utilized to print silkscreen text, visually labeling certain pin connections. Silkscreen labels were used to identify transceiver orientation, along with distinguishing which converter corresponds to what respective transceiver. Although not currently utilized, 2.62 mm diameter screw holes were included if any further design was to mount the PCB directly to a housing fixture.

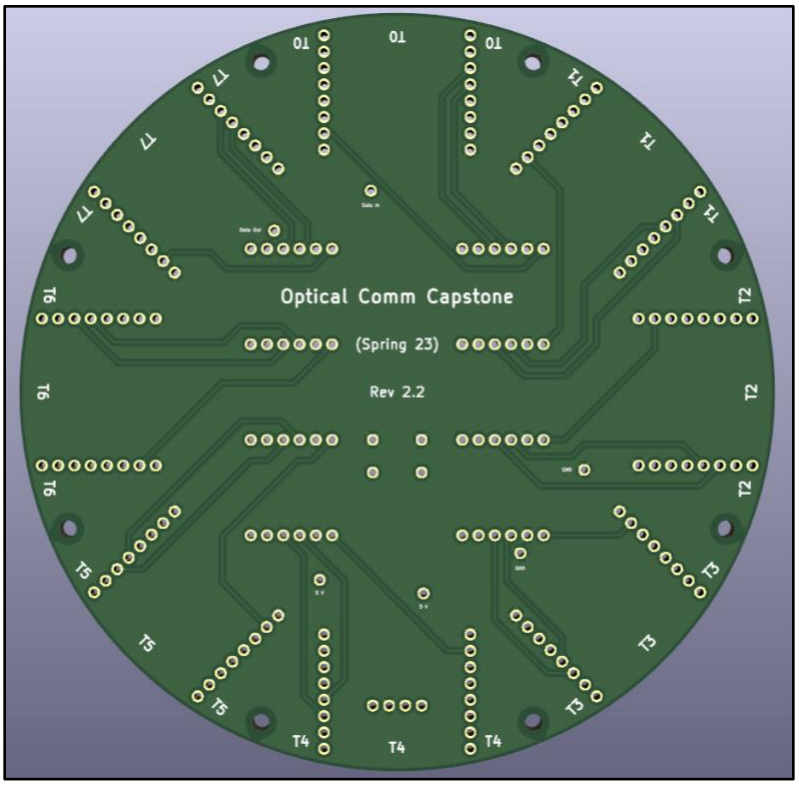

Figure 7: Current PCB Design (Revision 2.2), Top Side

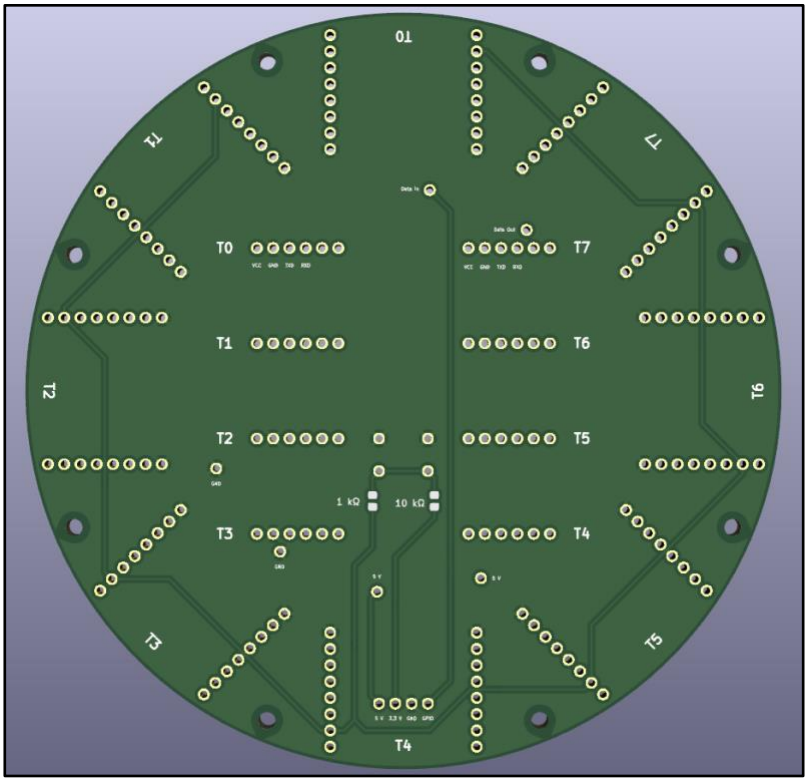

Figure 8: Current PCB Design (Revision 2.2), Bottom Side

3D Modeled Housing Fixtures

In order to fasten the PCB and other components to the mobile TurtleBot, 3D modeled housing fixtures were designed using the SOLIDWORKS software. Since the hardware components were assembled on the PCB, a unified structure was produced. Because of this, the overall design of the housing fixtures had to have been overhauled from the previous semester. A total of three parts were developed: Housing Base, Housing Lid, and the Device Tray. As seen in Figure 9, the housing fixture is assembled with the PCB and hardware components encased. Likewise, it can be noticed that the Device Tray is fastened behind the vertical spacers, and yet provides unobstructive access to the power bank and Raspberry Pi.

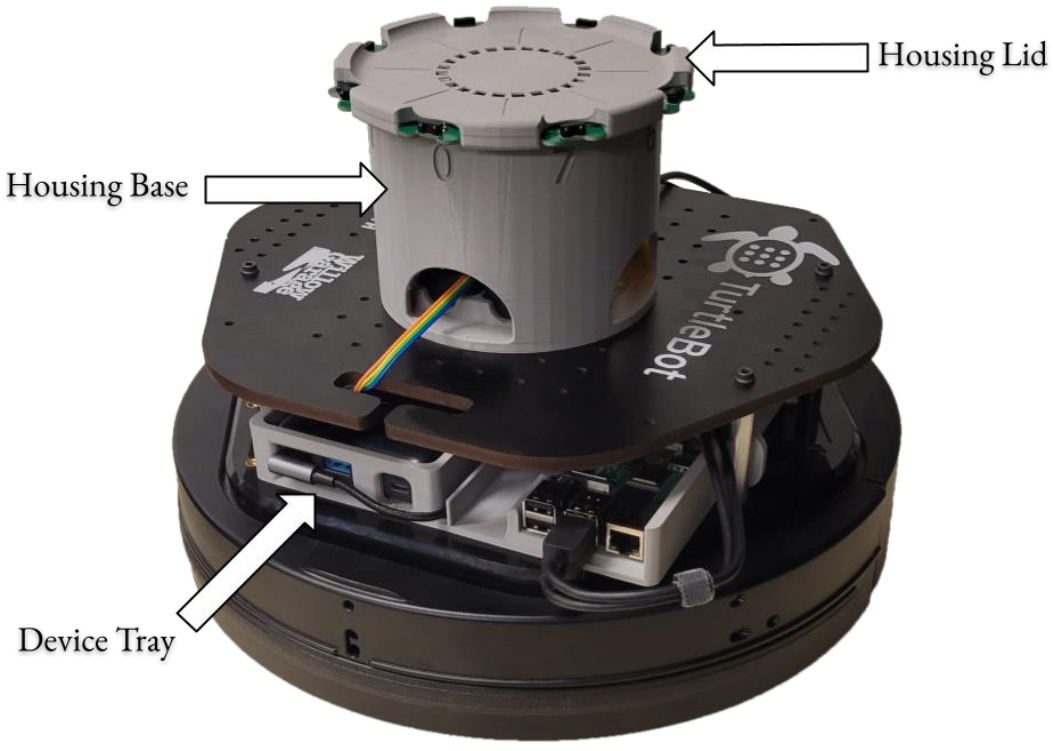

Figure 9: Final Assembly of Robot B

Housing Base:

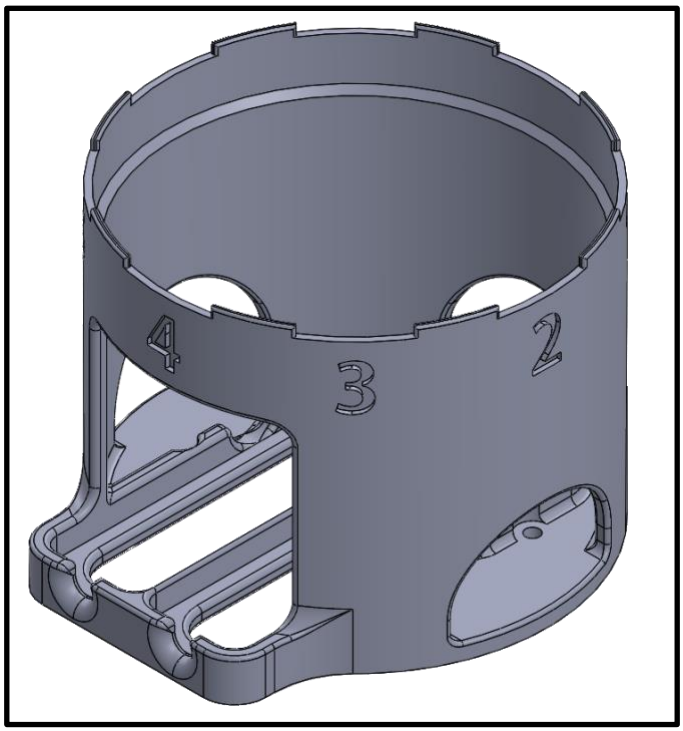

Figure 10: Housing Base (Mk 13)

The Base of the housing fixture was required to provide support to the PCB and to fix the hardware components to the mobile TurtleBot. Rather than producing multiple frame pieces that would stack on one another, this single Base design encloses the entire PCB structure. The floor of this design incorporates a raised foundation with 4 screw holes to allow the piece to be mounted to the TurtleBot's top shelf. When the PCB is inserted into the Base, the 2 USB hubs align with engraved divots that secure the USB hubs in place. The top of the Base fixture includes additional divots that the PCB's transceivers rest on. Under each resting ledge, an engraved label has been placed to number the transceivers. The manner in which the transceivers are ordered and numbered are consistent with the silkscreen labeling scheme on the PCB, as well as the numbered serial ports within the software. The external shell of the design offers easy access to the internal components through the implementation of curved access windows. These windows allow the operator to smoothly situate the PCB and USB hubs into the fixture Base, as well as improving the ability to remove the components during disassembly. This part file was printed on Medium speed, utilizing 15% infill.

Housing Lid:

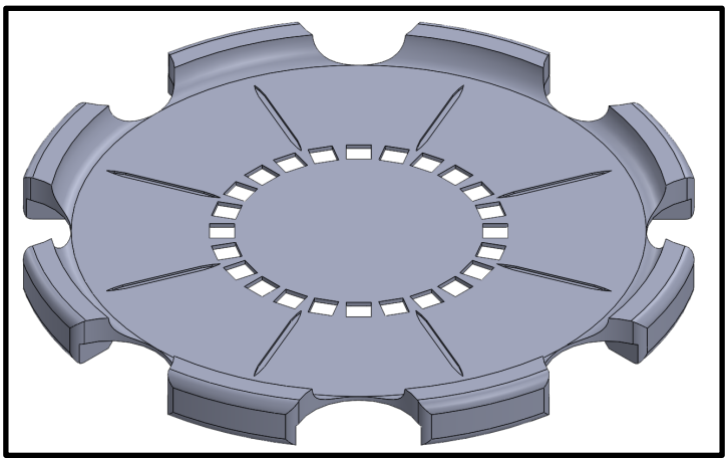

Figure 11: Housing Lid (Mk 5)

The Lid of the housing fixture sits upon the top of the Base piece, and slides into position with the help of chamfered sockets on the underside of the part. Because of the orientation of the transceiver's transmitter and receiver, it was possible for cross-talk to occur between adjacent devices. To prevent this, the edges of the Lid include thick walls that separate and shield the transceivers' line of sight from one another. Stationed in the center of the fixture's Lid are 24 square cutouts, accurately aligned with the position of the NeoPixel Ring on the PCB. These cutouts allow for the Ring's LEDs to remain visible, despite the rest of the PCB components becoming covered once the Lid is assembled. Because of the extremely thin framework, this part file was printed at Medium speed at 100% infill.

Device Tray:

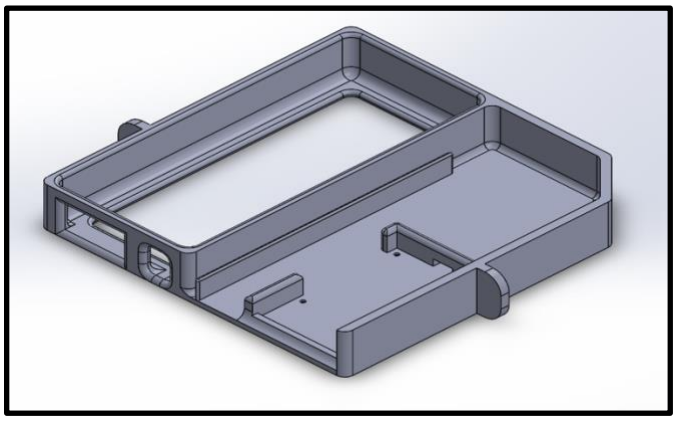

Figure 12: Device Tray (Mk 3)

A new addition to the robot's layout this semester is the Device Tray. This container securely houses the Raspberry Pi and its respective power bank underneath the TurtleBot's top shelf. This Tray design includes screw holes to safely mount the Raspberry Pi, along with a carefully measured wall to direct the Raspberry Pi's power cable to the battery in an organized fashion. It is worth noting that when deposited into the Tray, the battery's output and charging ports remain accessible to the user. Likewise, the Raspberry Pi's pinouts are easily reachable for any needed wire debugging. Located on either side of the Tray are handles that the user can utilize to slide the part in and out of position. For a more secure and permanent location, the handles can be situated behind the vertical spacers if the top shelf were to be temporarily removed. Because of the required strength of this container, this part file was printed at Medium speed with 30% infill.

V. Software Overview

Description

The system consists of a Discovery phase and a Maintenance phase. In the Discovery phase, the robots search for each other. If a pair of robots successfully discover each other, a Robot Link is created, which represents the new connection. The Discovery phase still continues to search for other robots, but it will not attempt to establish more connections with an already found robot. Once a new connection or Robot Link is established, the Discovery phase ends for finding that specific robot, and the Maintenance phase begins. The newly established Robot Link is maintained through a special Maintenance algorithm. The Maintenance phase is responsible for updating the transceiver used for sending the payload to the other robot. As both robots move around, a transceiver that was previously ideal for communication may no longer be in the line of sight of the other robot. Through the Maintenance phase, the best transceiver to use for communicating with the other robot is determined. The Maintenance phase only ends once a robot decides it no longer cares for communicating with a recipient robot, and the Robot Link can be discarded.

Architecture

Properly managing the incoming and outgoing data of the robots is essential to ensure that packets are not lost on the application layer, so that the Discovery and Maintenance algorithms work properly. Packets from other robots are first processed on Transceiver Receive Threads, which enqueue those received packets on a global data received queue. A Receive Manager Thread safely dequeues packets from the global data received queue, and writes them to the virtual serial port. Data that is written to the virtual serial port goes to the operating system, which may finally give that data to the appropriate sockets.

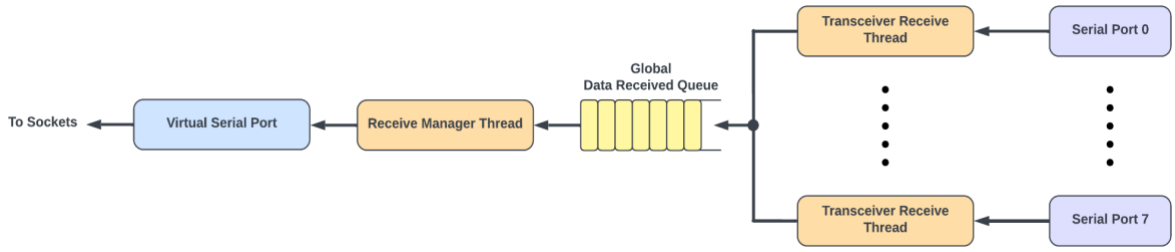

Figure 13: Data Receive

Packets that originate from the sockets are written to a virtual serial port. The Send Manager Thread reads the packets from the virtual serial port, and enqueues the packets on the appropriate Send Data Queue. For example, payload packets are enqueued on the Send Data Queue that corresponds to the best transceiver selected by the Maintenance algorithm. Each Transceiver Send Thread dequeues packets from a Send Data Queue, and then writes the packets to its serial port to have the transceiver transmit the data.

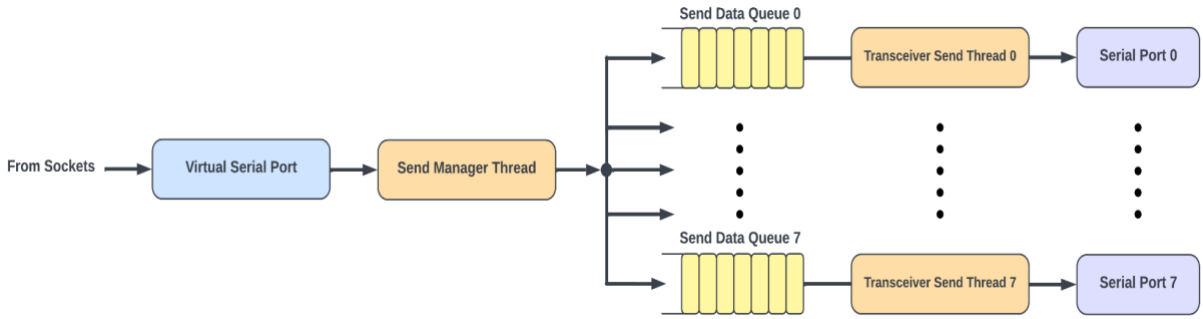

Figure 14: Data Send

For each Robot Link, the Link Send Thread utilizes a TCP socket to send a payload. When the socket sends data, it goes to the operating system, which then writes it to the virtual serial port. As depicted in Figure 14, data that is in the virtual serial port is read, and then sent through the appropriate transceiver. Also for each Robot Link, there is a Link Receive Thread, which reads received data from the TCP socket. As shown in Figure 13, when the Receive Manager Thread dequeues data from the global data received queue, it writes the data to the virtual serial port. Once data is written to the virtual serial port, the operating system may give the data to the TCP socket, where it is read on the Link Receive Thread.

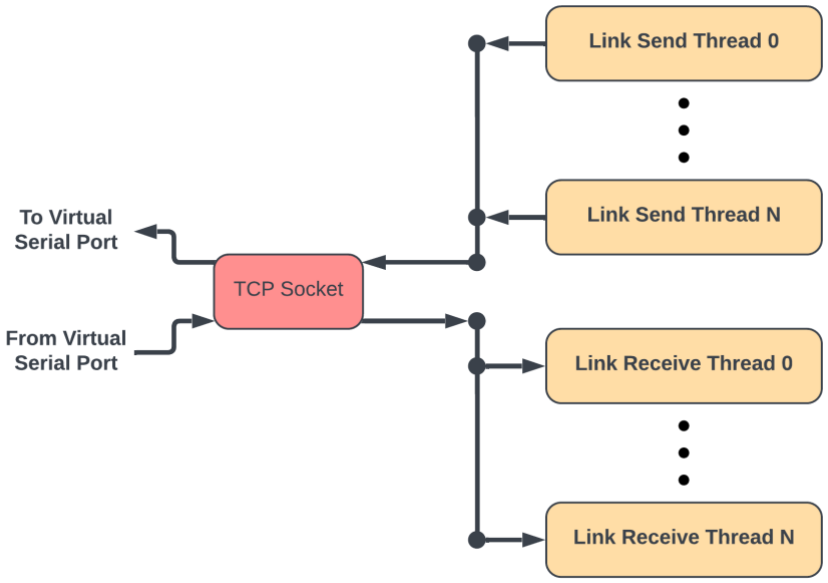

Figure 15: Link Send and Receive using TCP

For each Robot Link's Maintenance Thread, an ICMP Socket is created, which is used to send ping requests and receive ping replies, to determine the best transceiver to use for transmitting the payload. Once a ping request is sent using the ICMP socket, the ping request packet goes to the operating system, which then writes it to the virtual serial port. When ping replies are received, they are written to the virtual serial port. The operating system takes the ping reply packets from the virtual serial port, and gives them to the ICMP socket.

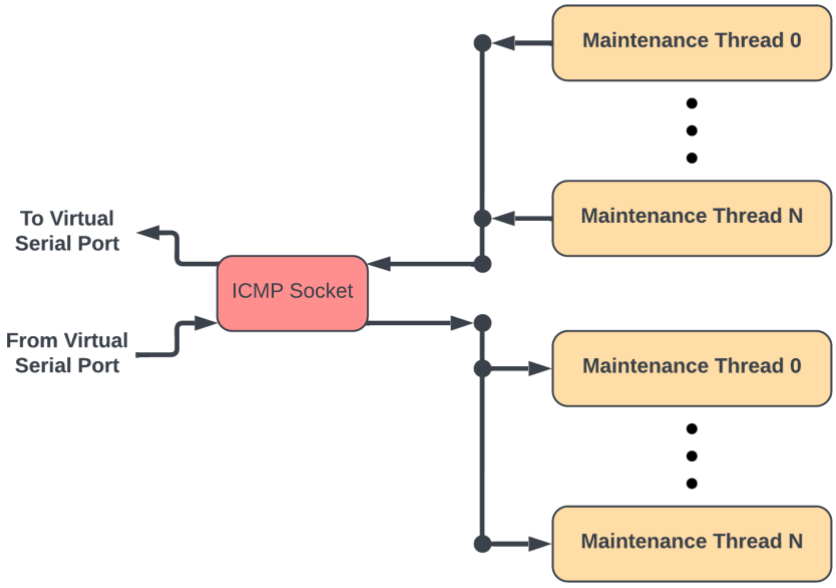

Figure 16: Maintenance using ICMP

Threading

Our project essentially requires extensive use of threading to simultaneously manage 8 different serial ports for sending and receiving data. Our implementation utilizes 20+ threads to manage communications between two robots alone, increasing for each additional robot. Significant consideration was needed to ensure that shared resources were not modified simultaneously. A queue module that implements a semaphore locking mechanism is widely used throughout the software as a safe shared resource. In addition to protecting shared resources, making the most of the limited CPU resources of our Raspberry Pi was also a top priority. Since there are 20+ threads, it is imperative that busy waiting is kept to a minimum, because busy waiting consumes CPU resources without actually performing any useful work. For every thread in the algorithm, there is at least one statement that blocks, or waits for a certain condition or event to occur before continuing. When a thread blocks, it releases the CPU, which can then focus on other threads that are not blocked. As a result, even though our software utilizes 20+ threads, it is possible that half or more of them are blocking at any given time, improving the system's overall efficiency.

VI. Software Final Design Implementation

Networking/Communication

Overall Network Design

The overall design of the network system uses sockets to communicate between the two robots. Transport Control Protocol (TCP) is used for the discovery process and to send the payload, and Internet Control Message Protocol (ICMP) is used for the maintenance process.

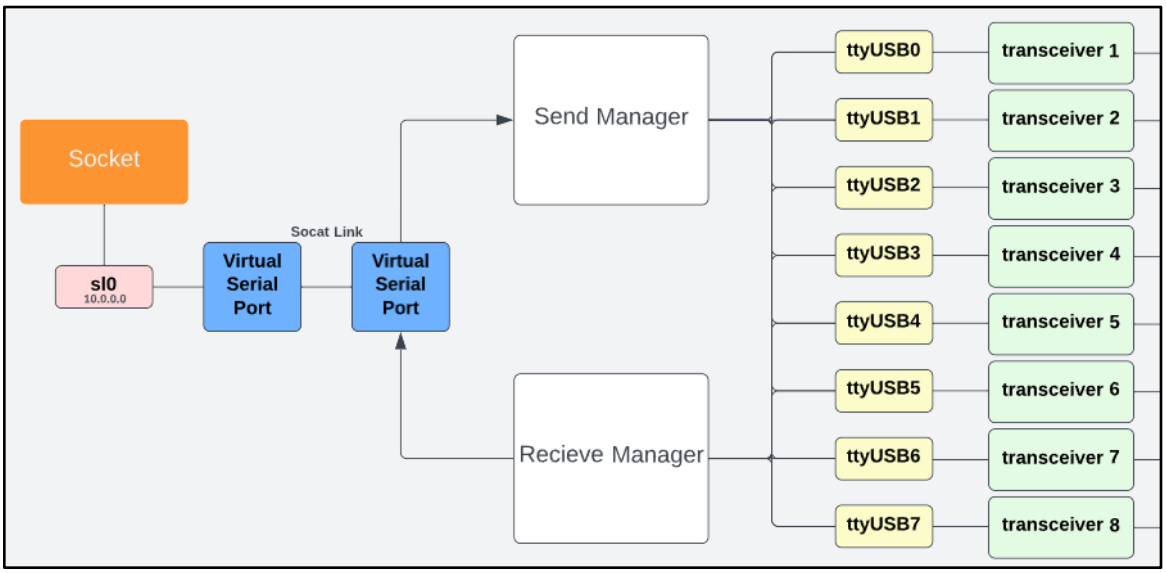

Figure 17: Network Diagram

Communication between robots systems is accomplished with a unique combination of existing network interfaces and protocols. **Serial Line Internet Protocol (SLIP)** is used to create **serial line interfaces** that can interact with the infrared transceivers. These interfaces function similarly to regular network interfaces, and they are created by a BASH command in our startup script. The interface that is created on each machine is associated with a **Socat Virtual Serial Port**. All data that is passed into the interface is forwarded to the serial port, and all data that is received by the serial port is forwarded to the interface. Python programs have been created (Send Manager and Receive Manager) to route data between a **Socat Virtual Serial Port** and the serial port that is attached to a specific transceiver. Once

the data enters the serial port for a transceiver, it is converted to infrared light and sent. When the light is received by the other robot, it will be converted back into serial data and processed.

After the network interface has been created, a socket is attached to the interface and used to send data as if a traditional wireless (RF) connection was being used. Data will be passed into the socket, packetized, and routed to one of the serial ports attached to a transceiver. The method for determining which transceiver the packet is done by our algorithms, and it will be discussed in detail throughout the rest of the Software Design Implementation section.

Socat Virtual Serial Bridge

Socat is a command line based utility that establishes two bidirectional byte streams and transfers data between them [2]. This utility is used in the software to create a virtual serial bridge, which is constructed out of two serial ports. When a serial bridge is created, the two serial ports will look something like this: **/dev/pts/1** and **/dev/pts/2** . All data that is passed into one of the serial ports is received by the other. For example, if "hello" is written to **/dev/pts/1** , it will be able to be read from **/dev/pts/2** .

This utility provides the ability to change the serial port attached to the serial line interface in real-time and eliminates the need to shutdown the interface. If this method was not used, a new serial line interface would need to be created every time transceivers were changed, and the same TCP socket would not be able to be used for the entire communication process.

Serial Line Internet Protocol (SLIP)

Serial Line Internet Protocol is a packet framing protocol that defines a sequence of characters to frame IP packets on a serial line [4]. By using this protocol we are able to utilize Internet Protocol (IP) across our serial communication, which allows us to use a reliable communication protocol such as Transport Control Protocol (TCP). The network interfaces created by SLIP are Serial Line Interfaces. Each of these interfaces is attached to a serial port (specified when the interface is created), and all data that is

sent to the interface will be passed through its serial port. When creating these interfaces you assign them an IP address, and the IP address can be added to the routing table and used to create sockets. In our current design, we use TCP and ICMP sockets.

Transport Control Protocol (TCP)

Transport Control Protocol (TCP) is a communication protocol that provides a reliable, in-order, byte-stream service to applications [5]. The protocol handles things like error checking and congestion control, and it is a fundamental communication protocol for computer networks.

This protocol is used after creating the serial line interface. Once the interface is created on a device, a TCP socket is created and bound to the interface. This socket will attempt to connect to the TCP sockets on other robots by using a three-way handshake. After a connection is established through TCP, communication will begin.

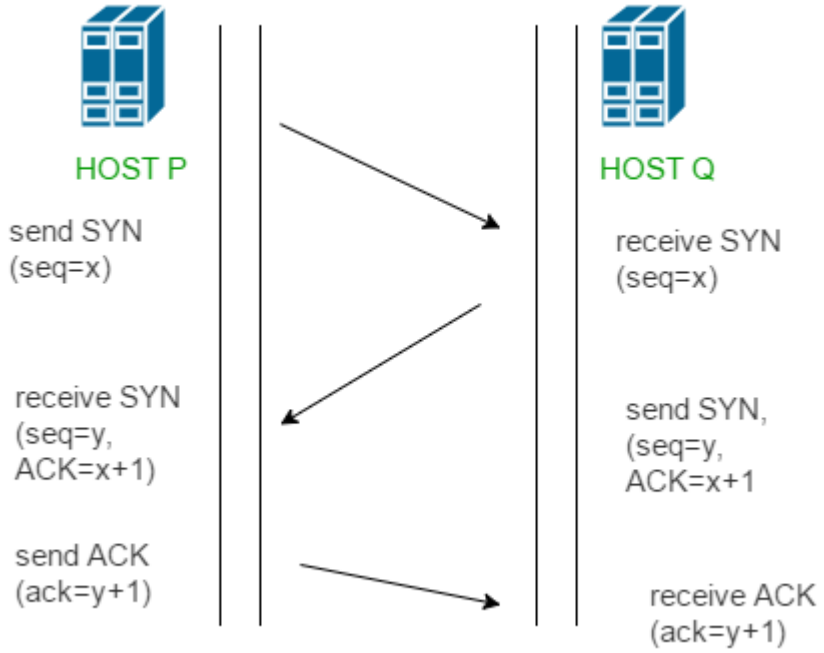

Figure 18: TCP 3-Way Handshake Process

Data is sent and received through this socket in the form of a byte array. Every TCP packet is given a header, which has information describing the packet. Using information from these headers, TCP ensures that all packets are received uncorrupted and in-order. If a packet contains corrupted data when it is received, the protocol automatically retransmits the packet. The header fields in the TCP packets are able to be read by the software, and they are used to route the payload packets between transceivers and the network interface. These header fields can be read by the software while the packets are passing through the physical layer of the OSI model.

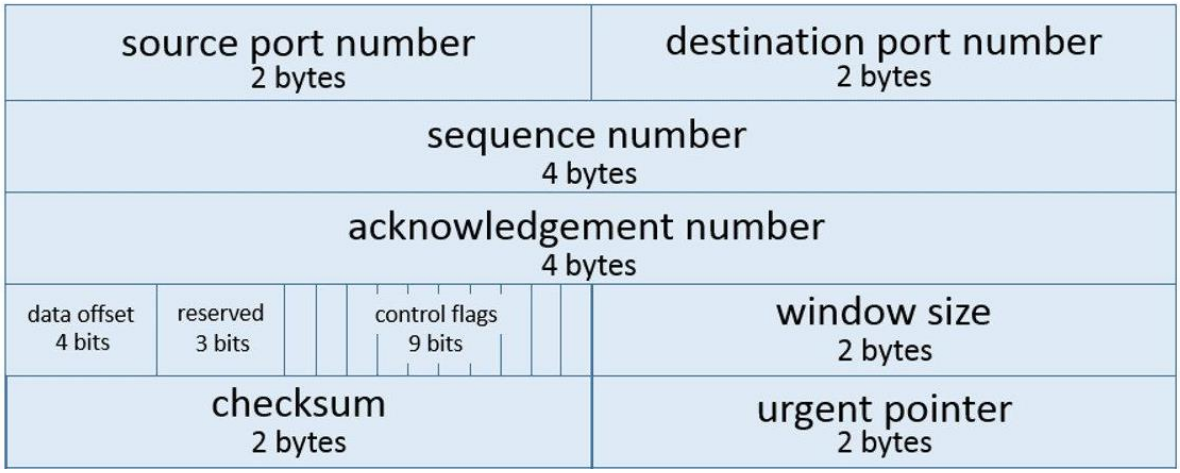

Figure 19: TCP Header

Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) is used by network devices to send operational information about the status of network communication. It is commonly used to troubleshoot issues in a network. In the system, ICMP is used in the maintenance process to determine the amount of packet loss in a connection. The system will always choose the transceiver that has the lowest amount of packet loss, and it will automatically switch between transceivers to use the strongest connection. Our system uses two main components from the ICMP header: the type field and the code field. These fields describe the purpose of the ICMP packet (reply or request).

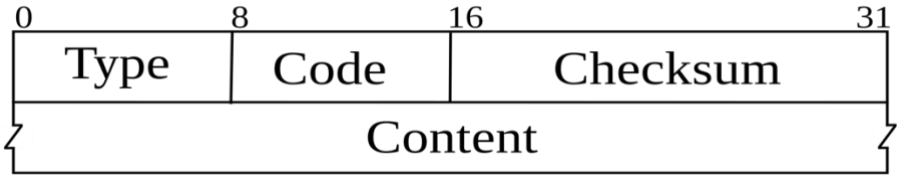

Figure 20: ICMP Header

Data Structures and Classes

Figure 21: Robot Link Class

The software utilizes classes as data structures for holding the associated attributes of an object together. Each robot link, or connection to another robot, includes that other robot's name, serial port, IP address, port, and socket. A robot link's serial port corresponds to the current transceiver that is used for transmitting the payload. The IP address and port of a robot link correspond to the IP address and port of the socket connection of the other robot. A robot link is established through discovery, and the maintenance algorithm updates the serial port to the one that provides the most reliable communication.

Transceiver Receive

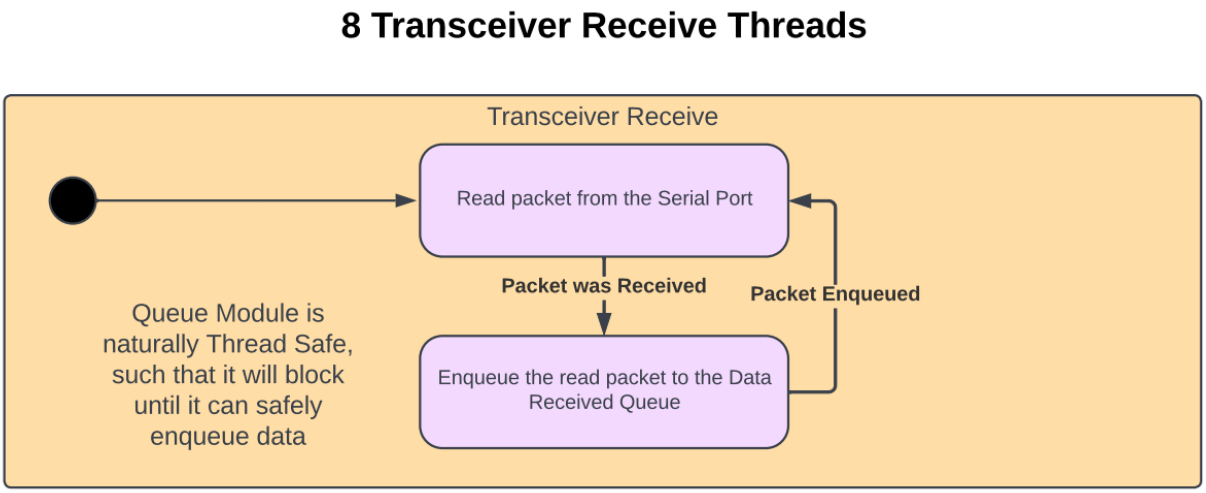

Figure 22: Transceiver Receive Thread

Each transceiver has a transceiver receive thread, which blocks until it is able to read a packet from its corresponding serial port. Once a packet has been received, it enqueues the data onto a global data received queue, which all transceivers receive threads share. The receive manager thread processes the received data on the global data received queue.

Transceiver Send

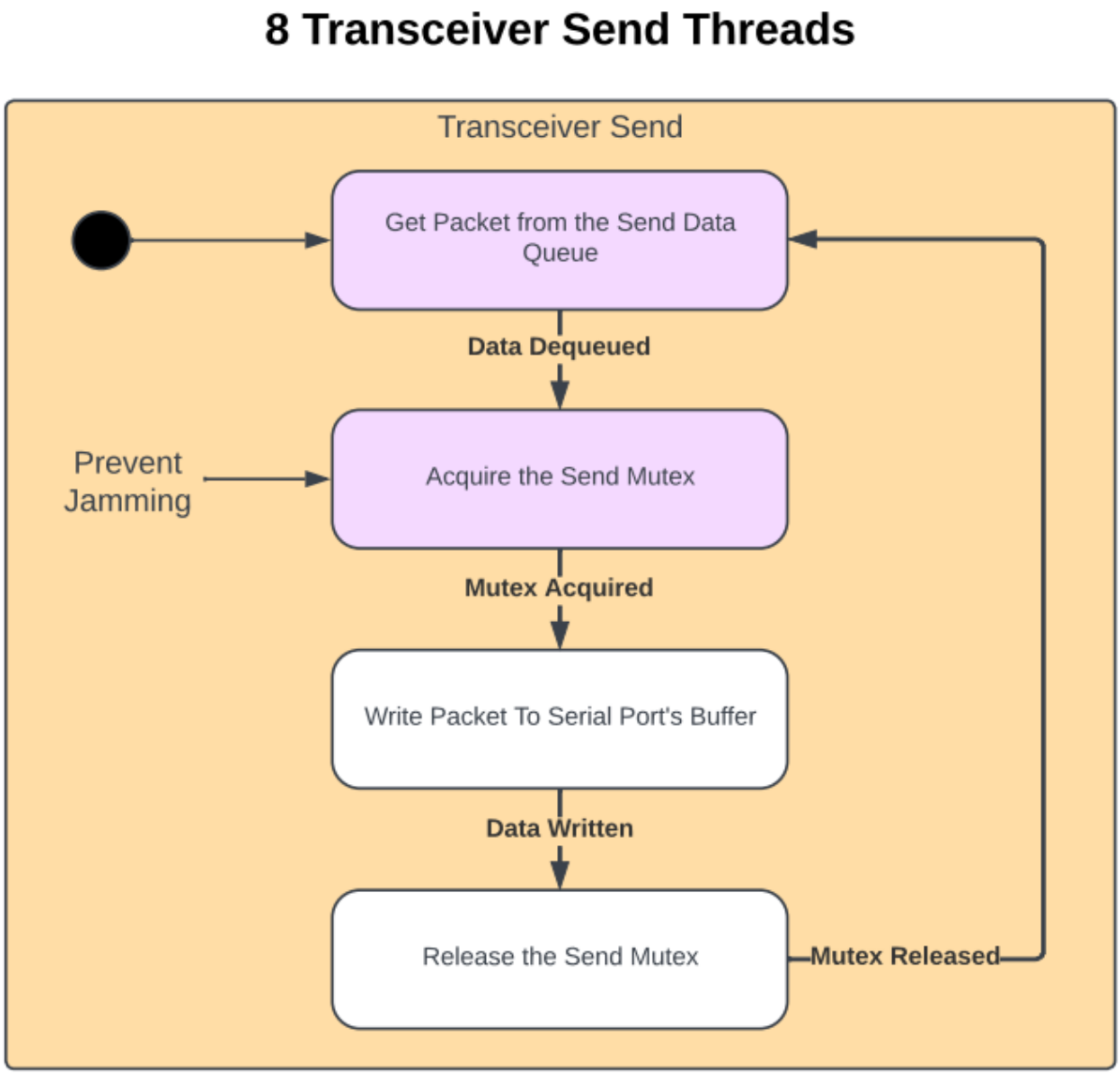

Figure 23: Transceiver Send Thread

Each transceiver has a transceiver send thread, and each transceiver send thread has its own send data queue. The thread blocks until it is able to dequeue a packet from its send data queue, and then it blocks until it is able to acquire a send mutex. The send mutex is used to prevent jamming in the event that multiple transceivers on the same robot would transmit at the same time because they share the same physical medium of the air. Once the packet is written to the serial port, the send mutex can be released, and the thread can block again until it has more data packets to send.

Receive Manager

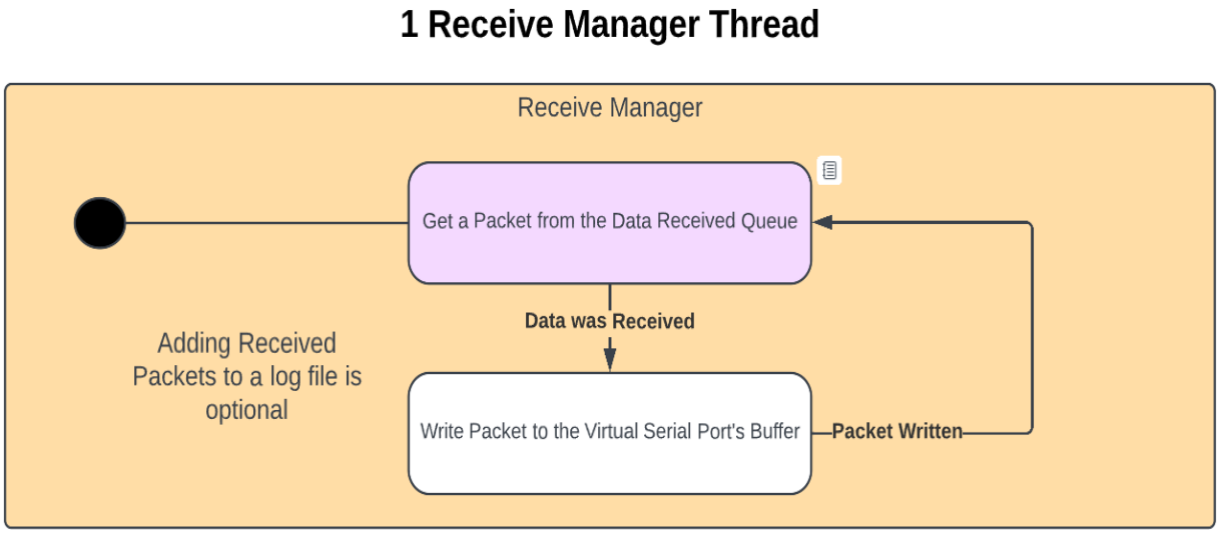

Figure 24: Receive Manager Thread

There is one receive manager thread, which dequeues received packets from a global data received queue that is populated by received data from the transceiver receive threads. Once a packet is dequeued, it writes the data to the virtual serial port, which is then processed by the operating system. The thread blocks until there is a packet to be dequeued from the global queue.

Send Manager

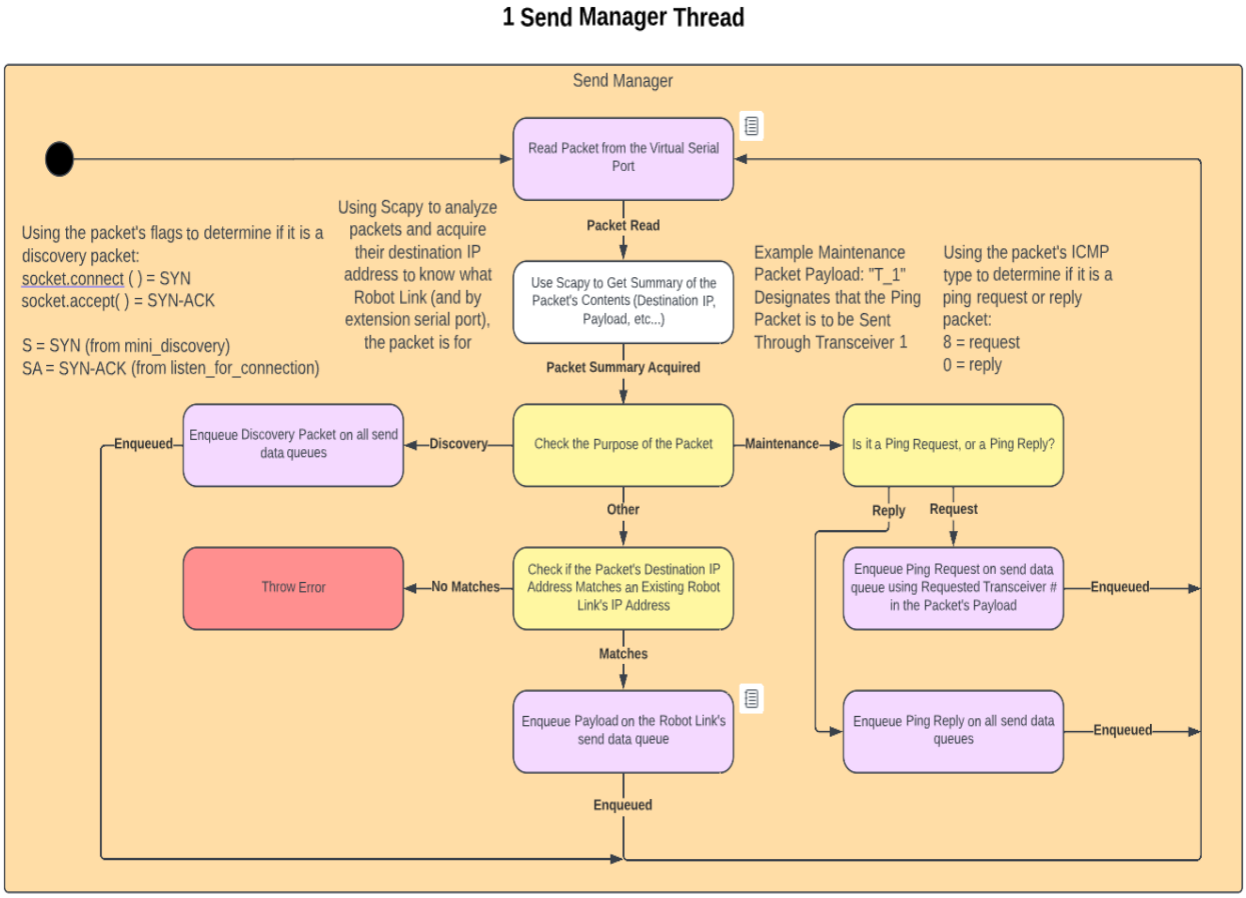

Figure 25: Send Manager Thread

There is one send manager thread, which blocks until it is able to read a packet from the virtual serial port. Once a packet is read, the python library Scapy is used to decode the packet and pull out its relevant information such as its destination IP address, payload, flags, and the packet's type. The purpose of the packet is checked, and if it is an ICMP or maintenance packet, it is checked for whether it is a ping request or ping reply using the ICMP type. Ping replies are put on all transceiver send data queues, and ping requests are put on a specific transceiver send data queue, using the requested transceiver number in the packet's payload. The purpose of the packet is determined to be discovery if it contains the TCP SYN or SYN-ACK flags. To discover other robots, the packets are put on all transceiver send data queues. Lastly, if it is neither a discovery or maintenance packet, the packet's destination IP address is used to determine which transceiver the payload should be sent on.

Link Send

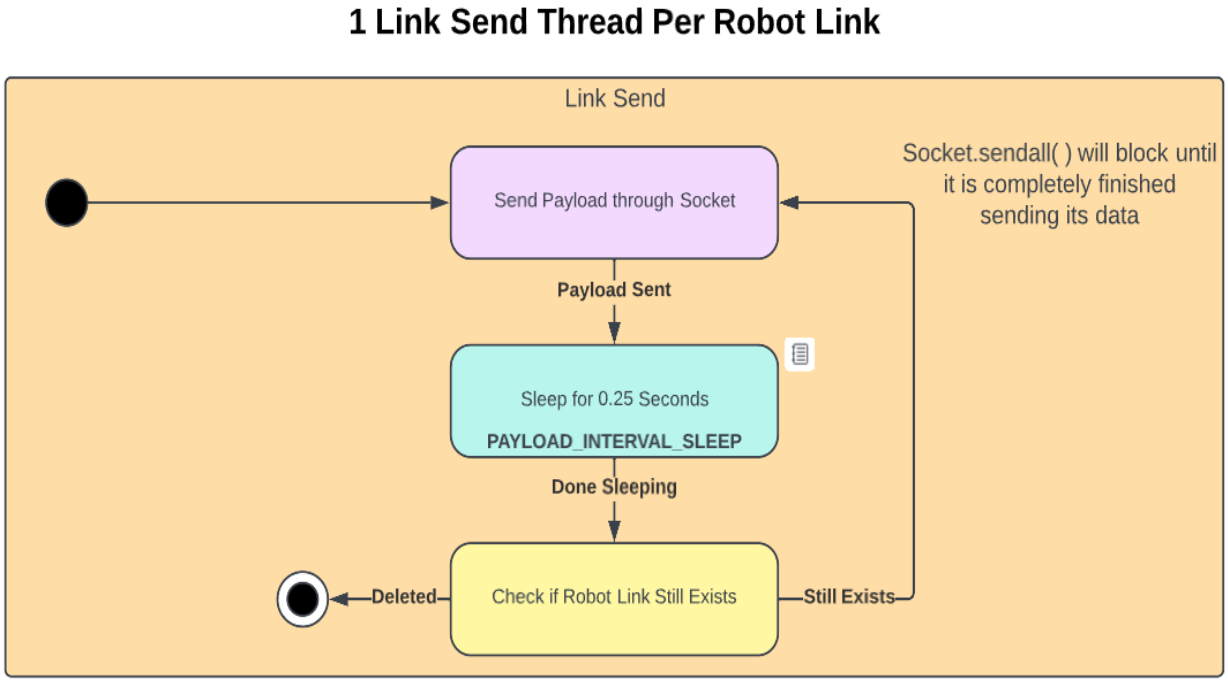

Figure 26: Link Send Thread

For each newly discovered robot, a link send thread is created through which a payload can be sent to the new robot if desired. Once the TCP socket sends the payload, the operating system gets the data and writes it to the virtual serial port. The send manager thread then reads the virtual serial port and gets the payload packet. Using the packet's destination IP address, the payload is sent through the transceiver that corresponds to the destination IP address of the robot link.

Link Receive

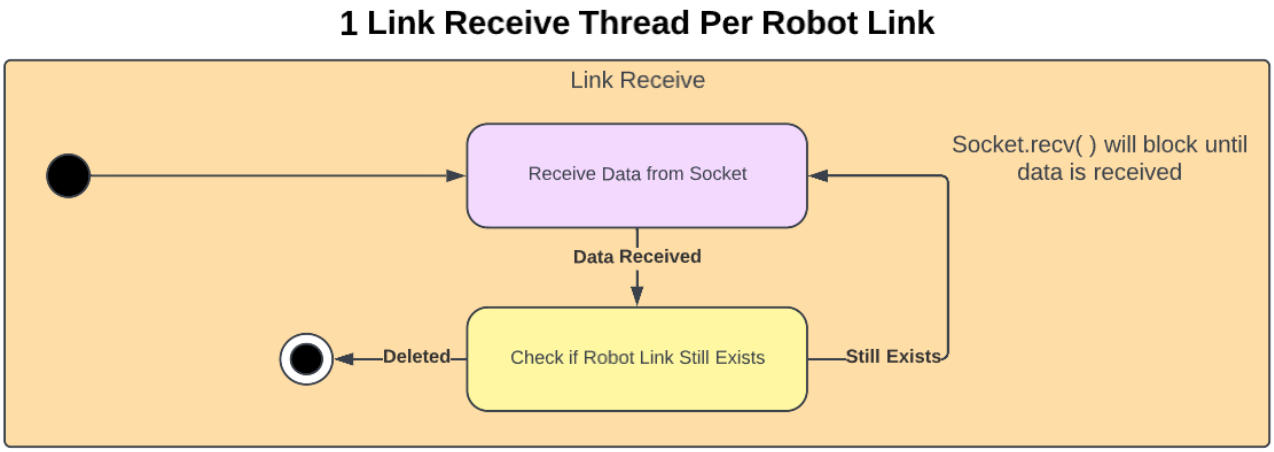

Figure 27: Link Receive Thread

For each newly discovered robot, a link receive thread is created, where data that was sent from the other robot is received. When the other robot sends a payload to the local robot, it is received in the transceiver receive threads, and then enqueued on the global data queue. The receive manager thread then writes the received packets to the virtual serial port's buffer, which gets processed by the operating system, and then finally received in the socket.

Discovery

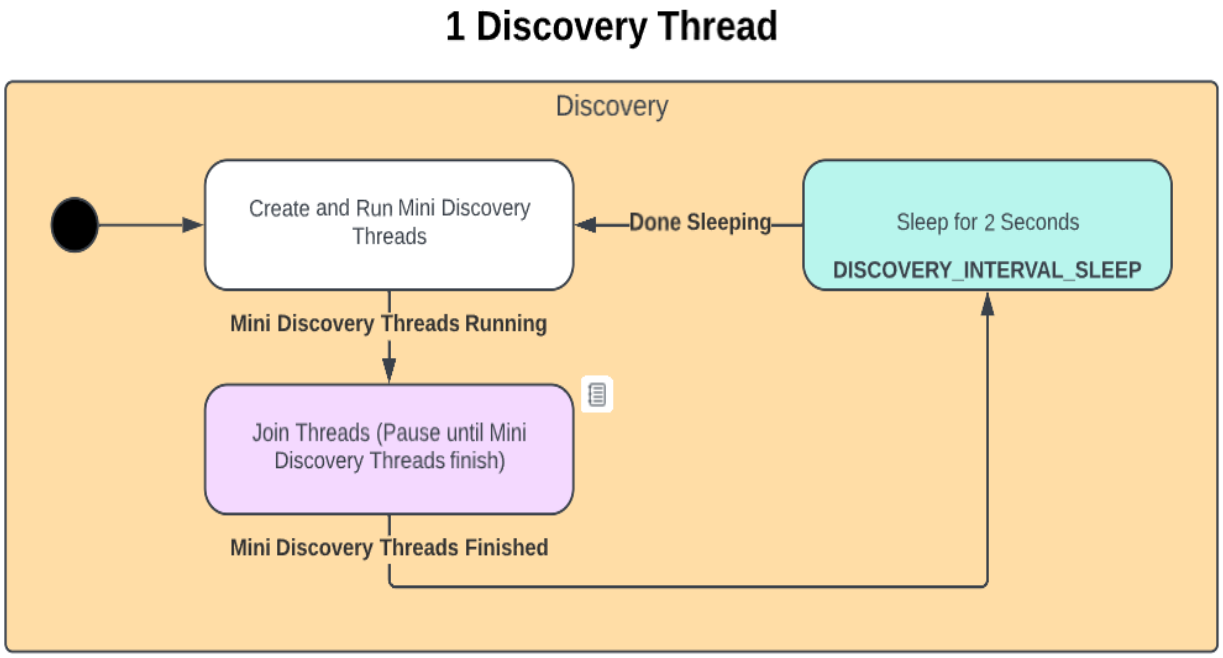

Figure 28: Discovery Thread

There is one discovery thread which is responsible for creating and running mini discovery threads on a regular basis to initiate connections with other robots. The discovery thread will block until all mini discovery threads finish, and then sleep for a specified constant.

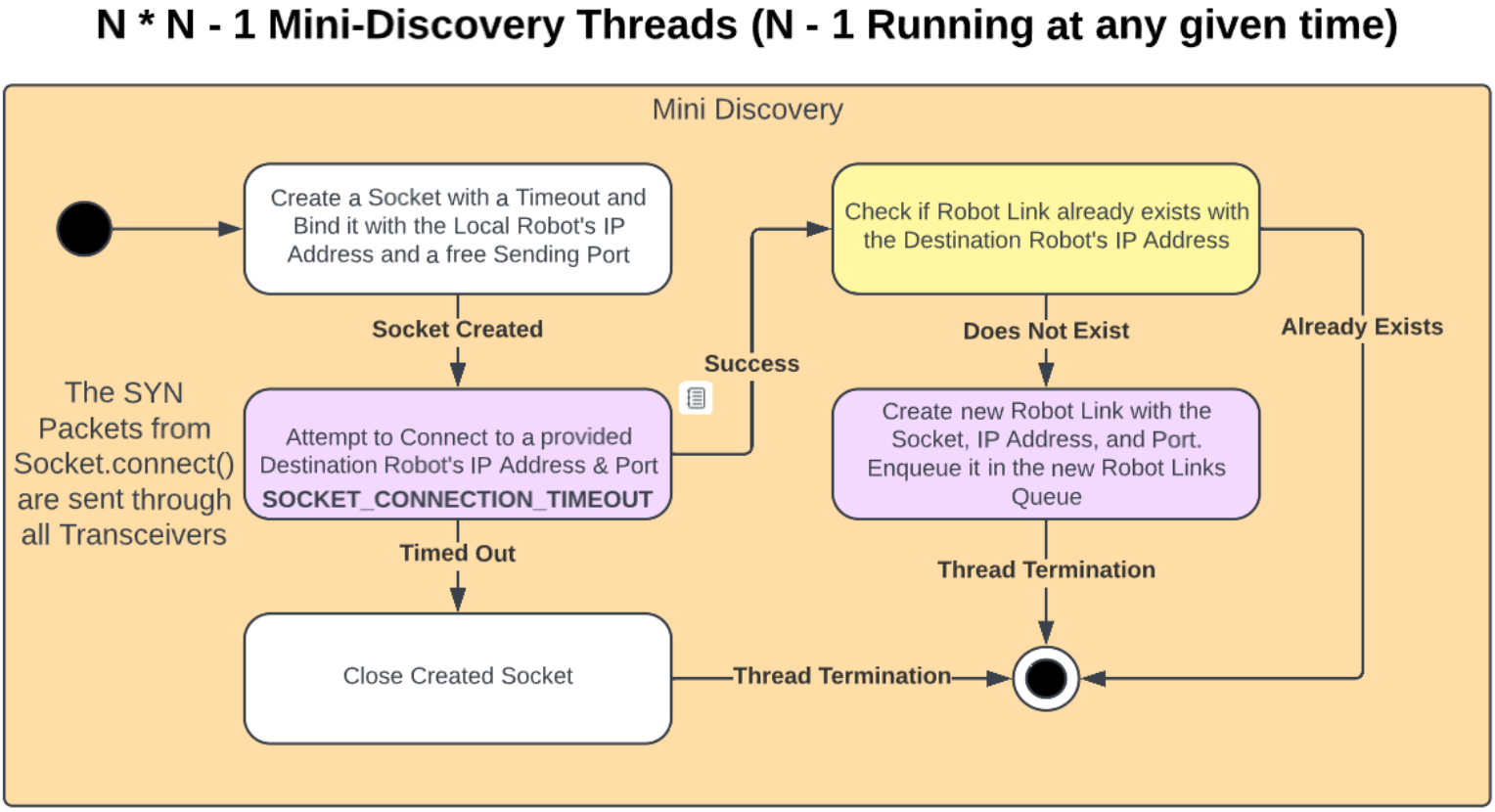

Figure 29: Mini Discovery Thread

The number of mini discovery threads running is dependent on the expected number of other robots. Each mini discovery thread creates a socket and binds it to the local robot's IP address, using a free port. Next, each mini discovery thread has a destination robot IP address and port combination that it attempts to connect to using a TCP socket. When the thread attempts to connect, SYN packets are created which the operating system writes to the virtual serial port. The send manager thread then reads the virtual serial port, and sends the SYN packets through all of the transceivers to discover other robots. If another robot is nearby and is listening using the IP address and port of the sent SYN packet, then it will reply by sending SYN-ACK packets back in response. Upon receiving SYN-ACK packets, the attempted socket connection will resolve successfully, and if a robot link with the same destination IP address does not already exist, a new robot link is created using the socket, IP address, and port. Otherwise, if no SYN-ACK packets are received, then the socket will timeout and be closed, followed by thread termination.

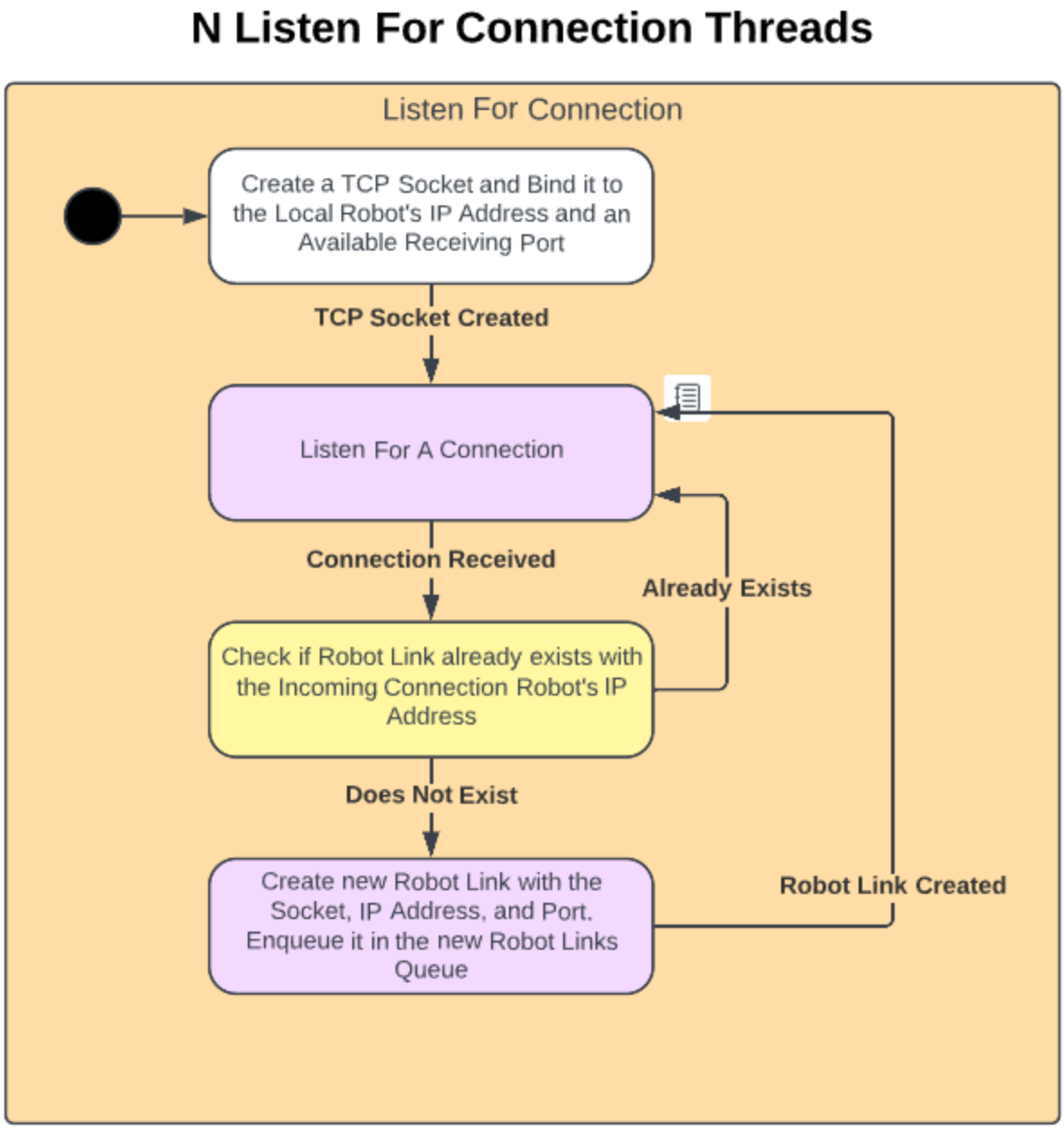

Figure 30: Listen For Connection Thread

The number of listen for connection threads running is dependent on the expected number of other robots. For each listen for connection thread, a TCP socket is created and bound to the local robot's IP address, and a free port. Next, the socket listens for connections, and replies with SYN-ACK packets if a connection is received. If a new connection originates from a robot that has no current connection, then a new robot link is created using the client connection's IP address, port, and the socket itself.

Maintenance

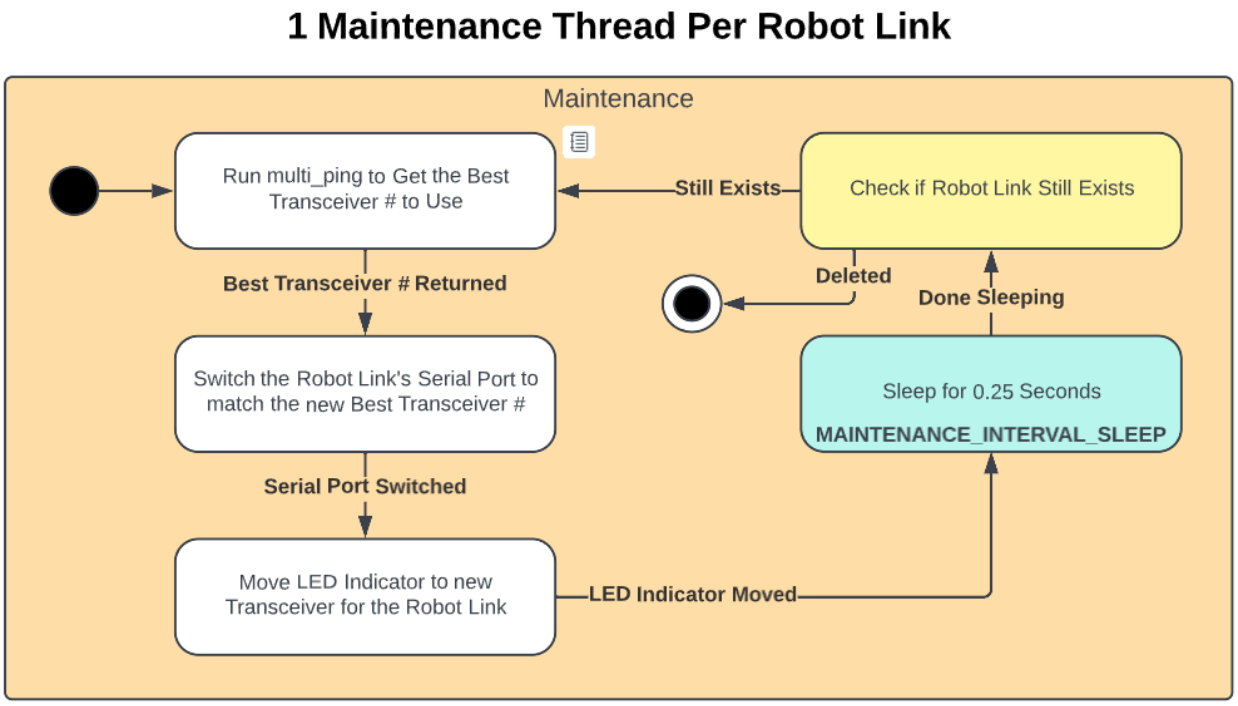

Figure 31: Maintenance Thread

For each newly discovered robot, a maintenance thread runs to update the serial port (and by extension the transceiver) to the best one for reliable communication with the other robot. A multi ping function is run which returns the best transceiver number (transceivers are numbered from 0 to 7) to use for communication. The multi ping function determines the best transceiver to use by creating a raw ICMP socket, and sending request pings with unique identifiers through specific transceivers. When another robot receives a request ping, it will send back reply pings with the same unique identifier that it received. The number of reply ping packets of a specific identifier indicates that the communication reliability of that transceiver with the other robot, where more reply ping packets received indicate a stronger connection. Once the ping function is finished sending and receiving pings, it returns the transceiver number that received the most reply pings that match its sent unique identifier. Upon getting the best transceiver number, the maintenance thread updates the robot link's serial port and LED indicator to match the new best transceiver number. Lastly, the thread sleeps for a specified constant, and reruns multi ping if the robot link still exists.

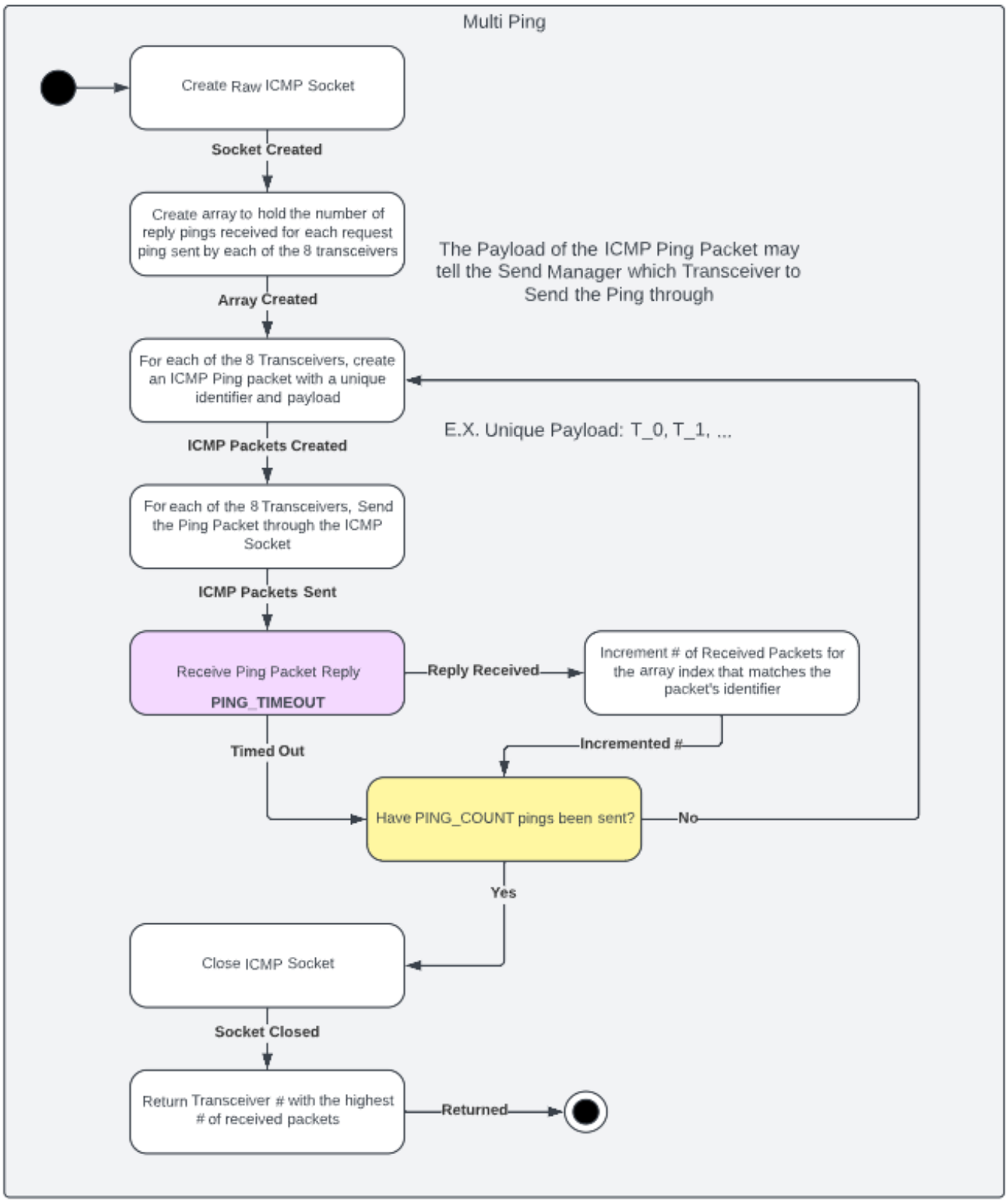

Figure 32: Multi Ping Function

VII. Analysis of Design

Two tests were done to analyze the performance of the communication system. The first test measured the reliability of the system at various distances by monitoring the packet loss rate as the robots were moved farther apart, and the second test monitored the average packet round trip time as the robots were moved farther apart.

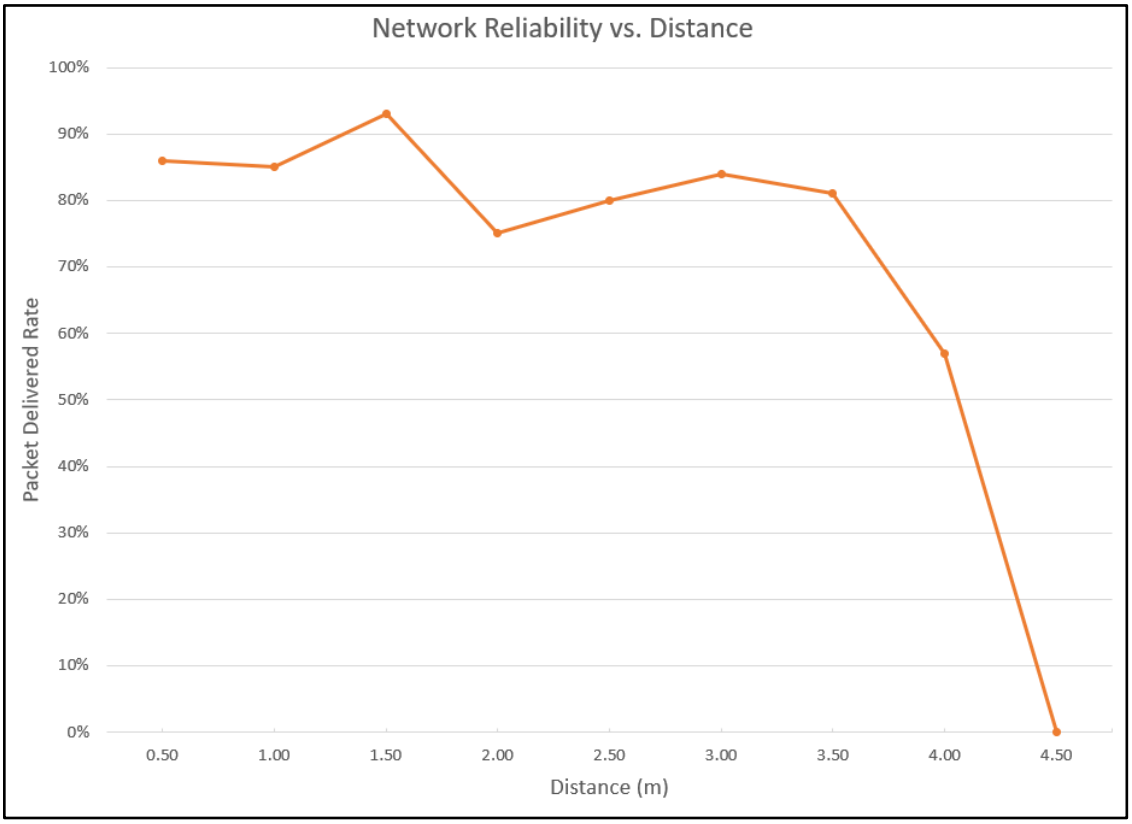

Figure 33. Packet Delivered Rate vs. Distance

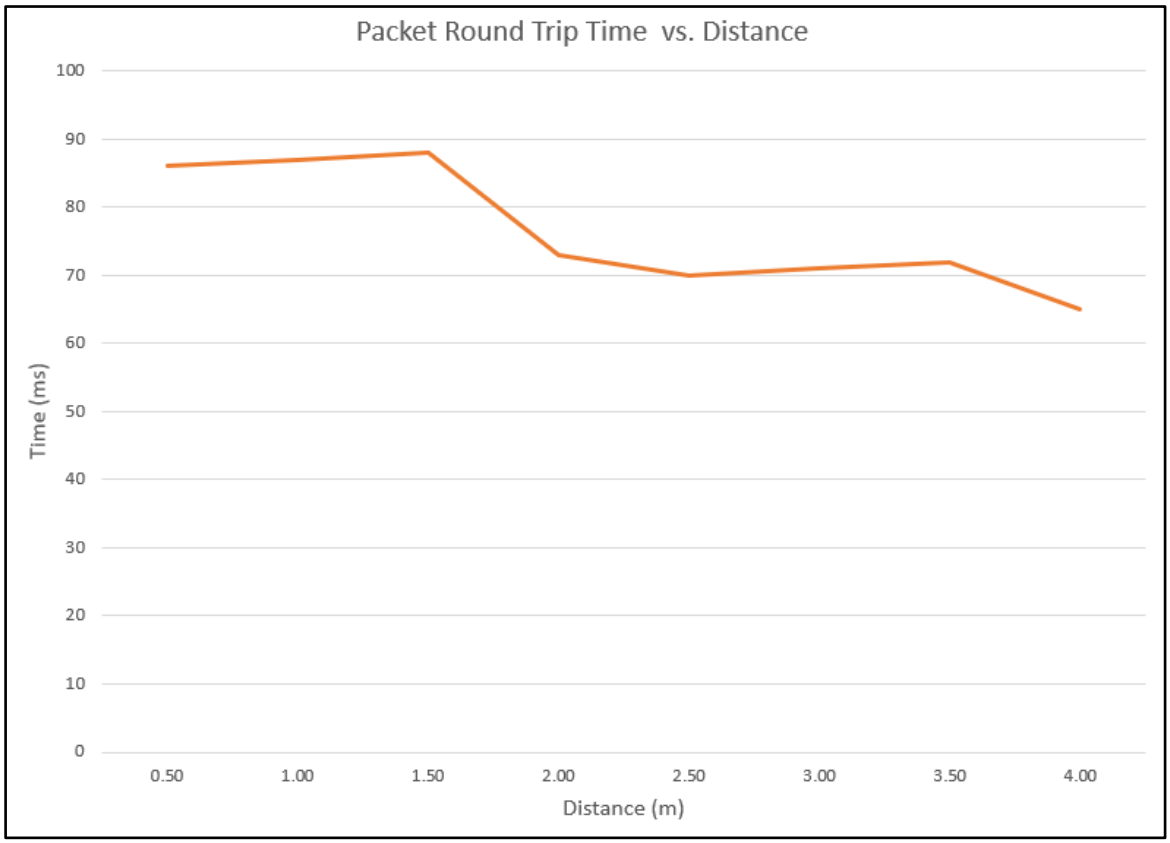

Figure 34. Packet Round Trip Time vs. Distance

From these tests, it can be concluded that the communication is most effective at distances less than 4 meters. Once the robots are more than 4 meters apart, the communication becomes ineffective and the robots are unable to establish a connection in the discovery phase. The transceivers used in this iteration of the project have a limitation of 4 meters, so these results are expected.

As the distance between the robots increases, the average packet round trip time decreases. As the distance increases, less packets are being successfully transmitted to the receiving robot. This reduces the number of packets that need to be processed by the robot, which decreases the time required for the software to run. Since the software is able to process the packets faster, it is able to send replies faster, which decreases the overall round trip time for the packets.

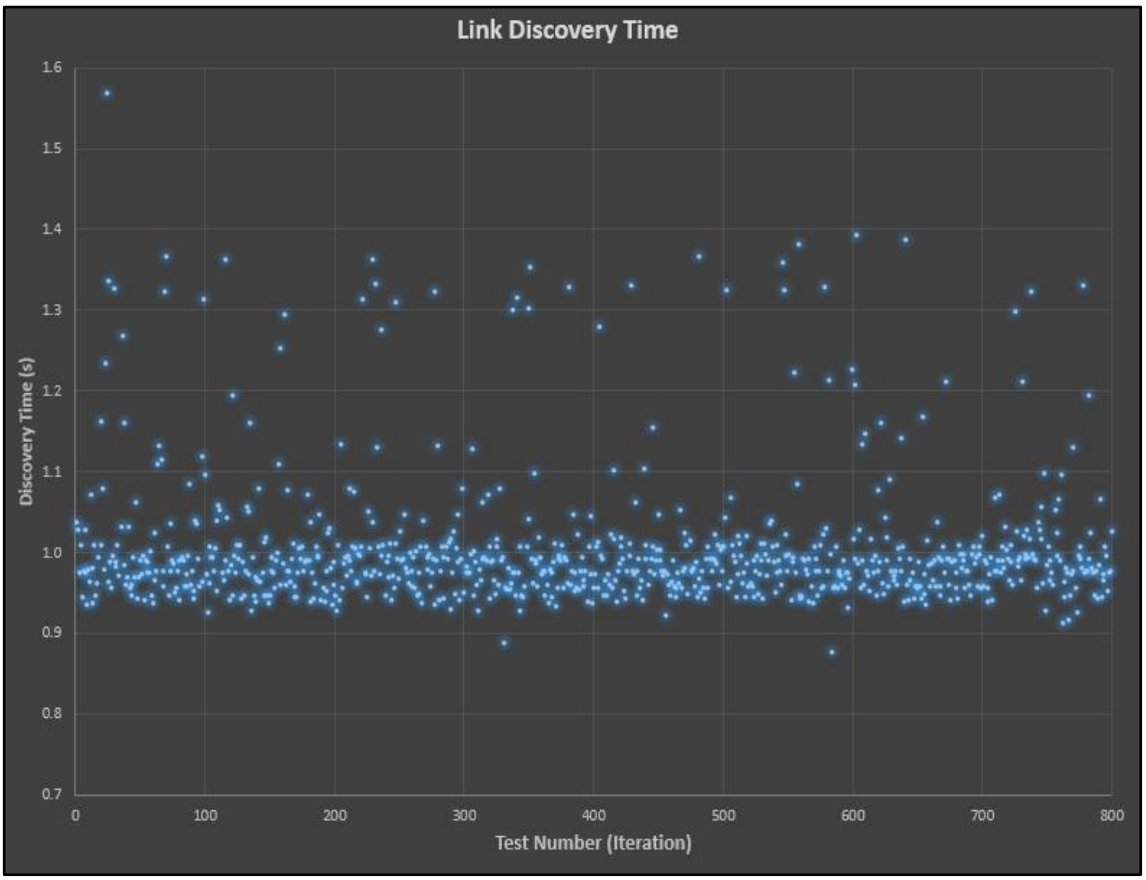

Figure 36. Link Discovery Time

The link discovery time refers to the time it takes for the robots to establish a good connection. In this process, the robots look for each other, create a network socket, and determine which of the transceivers will provide the best connection for optical communication. The time it takes for this discovery process to complete is highly variable, so 800 tests were conducted to obtain an accurate dataset. All of these tests were done in the same environment, and no variables (lighting, distance apart, etc.) were changed in between tests.

From the results shown in ***figure 36***, it can be seen that the robots take around 1 second to discover each other and begin data transmission (with a good connection). For the entire test, the average discovery time was **1.005 seconds** with a standard deviation of **0.084 seconds**.

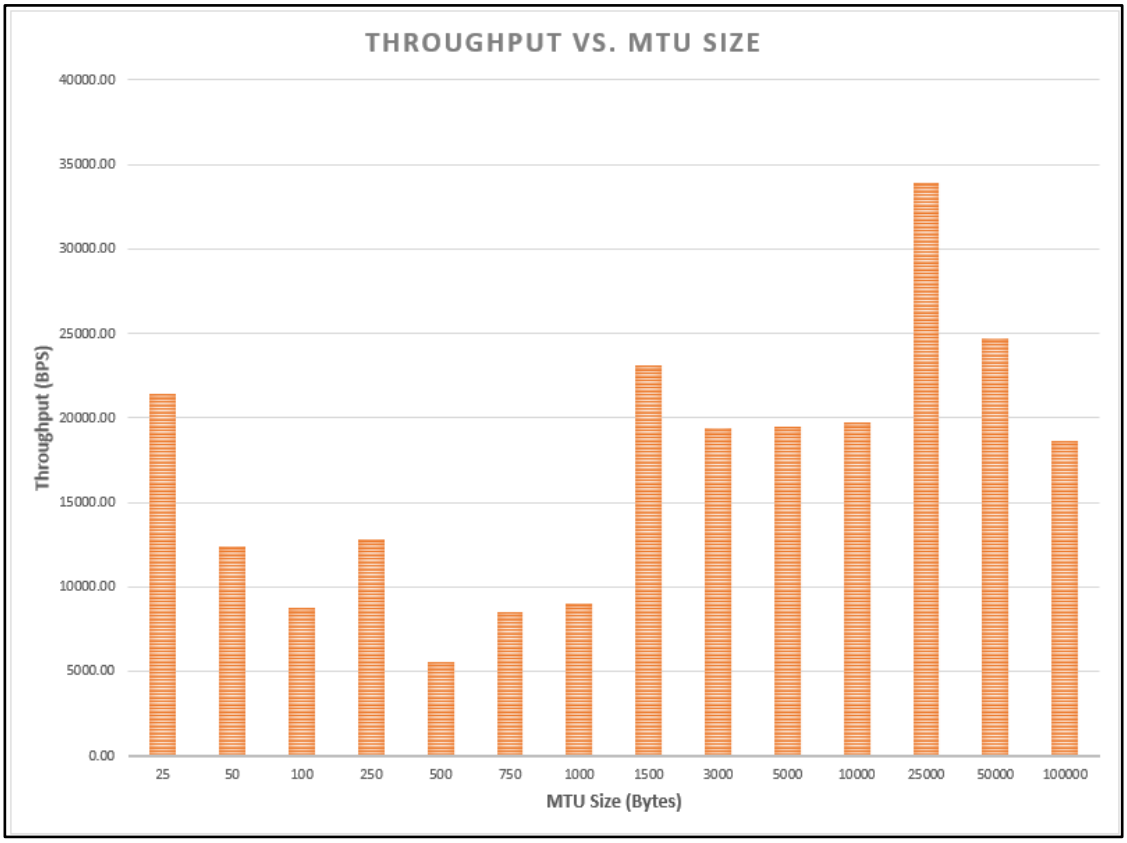

Figure 37. Throughput Rate for Various MTU Sizes

The throughput rate represents how much data we can send through our communication network (in bits per second). For these tests, the maximum transmission unit (MTU) size on the network interface was modified, and the throughput of the transmission was recorded. As the MTU size of the network interface is increased, the amount of data contained in each packet will increase. Having a larger packet size will reduce the number of packets needed to transmit data, but it will increase the chance that the packet becomes corrupted during transmission and needs to be retransmitted by TCP.

Overall, the largest throughput was recorded when using a large MTU size. As the MTU size was decreased, the average throughput of the system decreased. Additionally, there was an uptick in throughput when the MTU size was very small. From this data, it can be concluded that the optical communication system functions better with a larger MTU size.

VIII. Conclusion

Discovery/Maintenance With AI Image Recognition

Using AI image recognition, or computer vision, was considered as an option for both Discovery and Maintenance. In the current software, Discovery is used for creating the TCP socket connections between two robots, but not necessarily the transceiver used for sending the payload. Once the TCP socket connection is established, the Maintenance algorithm is what sets the transceiver to be used for sending the payload. It was previously thought that Discovery would only be for initially selecting the transceiver for communication, and Maintenance would be for updating the transceiver. Using AI Image Recognition, the Discovery algorithm could be having a camera turn using a motor until the AI is highly confident that another robot is in sight. The Maintenance algorithm could then be having the camera attempt to track the movement of the other robot. Based on the camera's position, a transceiver could be selected for communicating with the robot the camera is pointing at. Despite its potential upsides, there are some downsides to using a camera for Discovery and Maintenance. Our current software supports using N robots, or as many robots as the operating system can open ports (~60000). Using a camera may limit communications to only one robot at a time, because if there was only one camera, it may struggle to track more than one robot for maintenance. In addition, all robots may need to look the same, and the camera may begin tracking a different robot if two robots were two close to each other. All in all, using AI image recognition would pose new challenges for communicating with multiple robots.

IX. References

[1] G. Rieger, “Multipurpose relay (SOcket CAT),” Manpages.org, 2022. https://manpages.org/socat#description.

[2] J. Postel, “Internet Control Message Protocol,” Sep. 1981. https://doi.org/10.17487/rfc0792.

[3] J. Romkey, “Nonstandard for transmission of IP datagrams over serial lines: SLIP,” Jun. 1988. https://doi.org/10.17487/rfc1055.

[4] Pololu Corporation, “0.100″ (2.54 mm) Female Header: 1x8-Pin, Straight,” *Pololu.com*, 2023. https://www.pololu.com/product/1018.

[5] S. Sattel, “Top 10 PCB Routing Tips for Beginners | EAGLE | Blog,” *Fusion 360 Blog*, May 11, 2017. https://www.autodesk.com/products/fusion-360/blog/top-10-pcb-routing-tips-beginners/#:~:text=Avoid%20using%2090%2Ddegree%20trace,the%20copper%20on%20you%20board.

[6] W. M. Eddy, “RFC 9293: Transmission Control Protocol (TCP),” Ietf.org, Aug. 2022. https://www.ietf.org/rfc/rfc9293.html#name-changes-from-rfc-793.