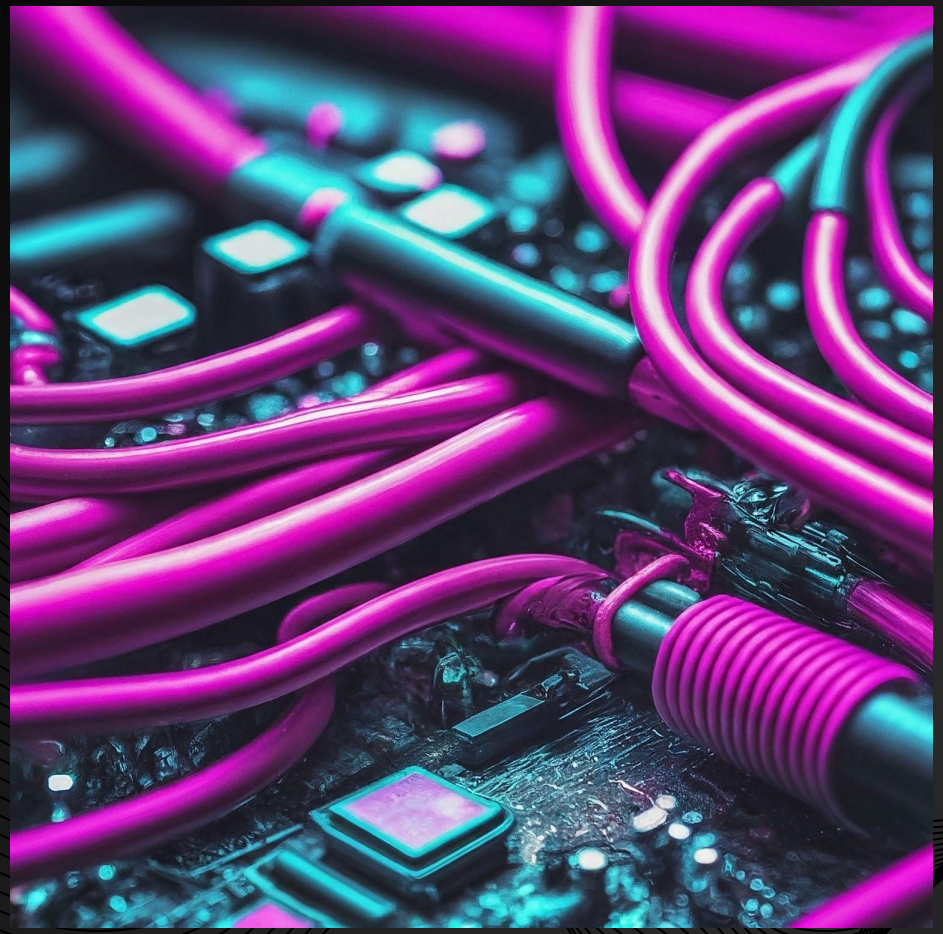
Robot Ethernet Framework

IP based module system



Systemic Issues

Brandee sub-teams struggled with sending data between systems.

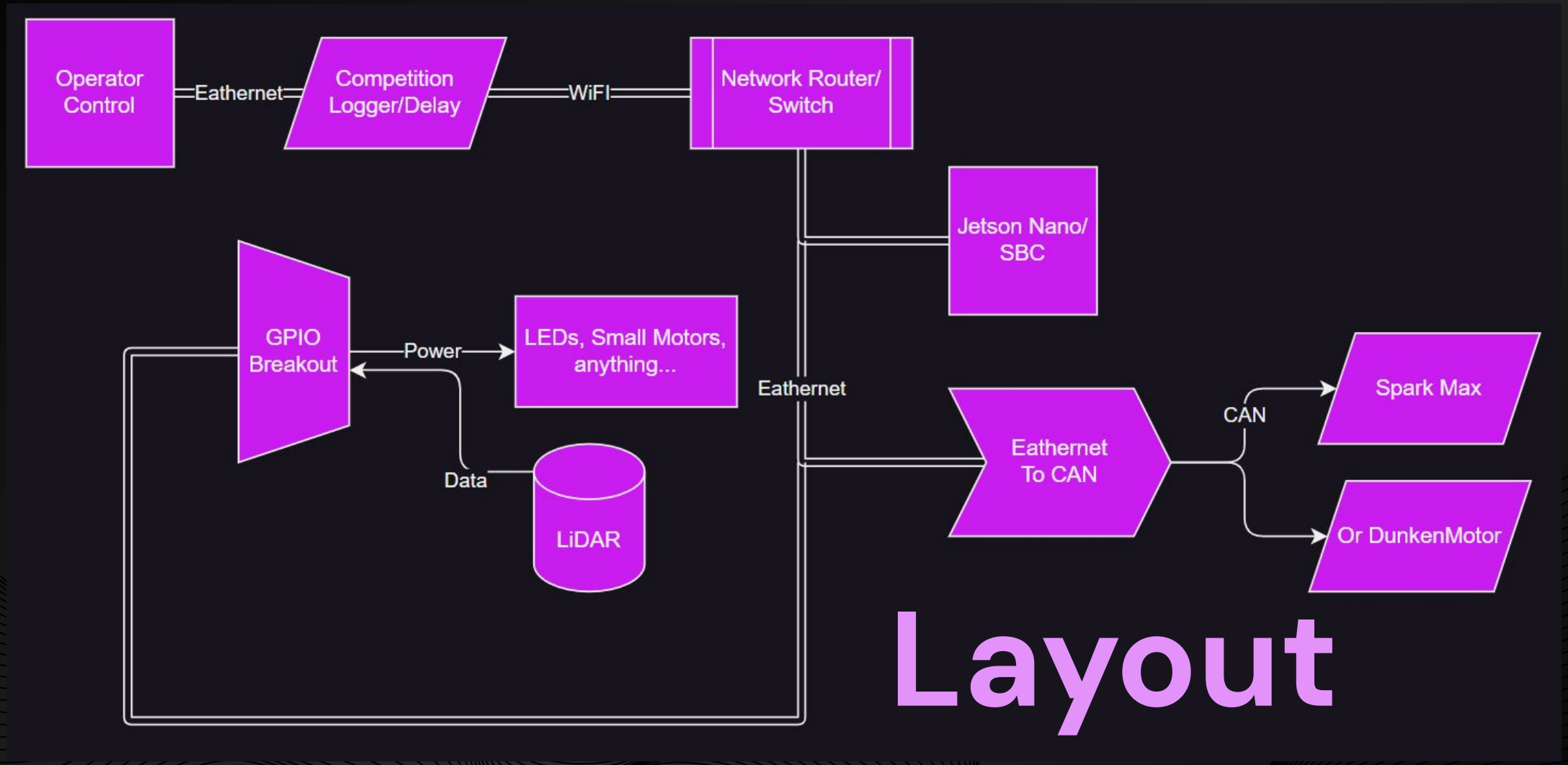
Arduino interface - Limit Jetson GPIO - Slow connections - Spark Max PWM/CAN

IP

Internet Protocol Communication Network

Use an onboard network to connect custom IO modules to the SBC and Operator Console.

Customizable modules allow for an easily adaptable framework.



Typical expected layout with a router/switch at the heart of the network. The network is flexible, and any additional IO modules needed can simply be plugged into the switch.

Comparing Speeds

GPIO	Ethernet
Jetson Nano Serial: 1,000,000 baud (1 client)	Jetson Nano Ethernet: 1,000,000,000 bits/s (No client or length restriction)
Jetson Nano I2c: 1,000,000 bits/s (Network < 4” in length, no interference protection)	The ethernet port has an expected bandwidth of 1000 Mb/s.
Expected combined bandwidth 2-450Mb/s	

Interconnection

Interconnection allows modules to communicate between themselves, and dynamically switch control if components fail.

Adds possibility for redundancy to be implemented. Failover to a second SBC, or to the operator console

Other networks, minus CAN, are only a slave to master only communication.

Remote Modules

Possible using POE

A standard POE switch can transmit up to 200W of total power over the ethernet. This power can be harvested by components for use in low power applications like a GPIO module, or a LiDAR controller.

The benefit is power and communication over one cable, a simple setup that is easy to modify as the robot changes.

Board Brainstorm

01 - GPIO +

General IO Expansion for

Nano, IO replacement.

02 - To CAN

A dedicated ethernet to CAN adaptor.

03 - Actuator

H Bridge motor controller

with built in comms.

04 - Lidar

Lidar speed controller and communications board.

05 - Data In

Data collection module, with inputs from sensors.

06 - Accessory

Other boards, such as a RGB led driver.

Module Requirements

Microcontroller Power

To process the IP traffic into actions, we need some microcontroller. I would promote the use of the ATMEGA328P, the chip powering the Arduino Nano. It's a small but powerful chip that is affordable and well maintained. Other options include the PIC20 that the team used to use.

DIP Switch IPv4 Final Byte Selection

Each device needs its own IP, and rather than deal with some sort of DHCP and DNS, it makes sense to assign a static IP to every device, and set the host address octet by dip-switch so new firmware isn't needed.

Ethernet ASIC/POE Harvester

The WIZ5500 chip is a great ethernet to serial conversion chip to allow the ATMEGA328P to access the network, and an isolation transformer allows POE to be pulled out of the signals ahead of the WIZ5500.

Communications Standard

Give a man a fish, he eats for a day. But teach a man how to use websockets, and he can order food onli

George Washington

3 Primary Websocket Ports

57344

TCP, Polling by SBC to each module. Read/Write IO, general data transfer (2^16-2^13)

57600

UDP, Interrupt Service Port. Alerts from modules that can’t wait to be polled. (2^16-3^13+2^8)

57664

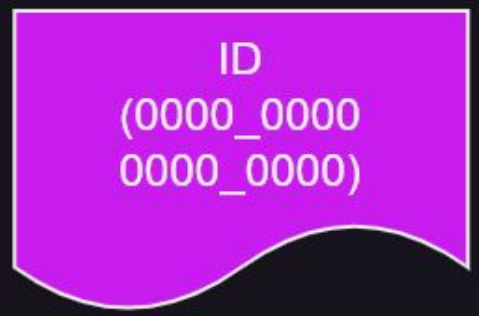
TCP, System Admin. Status, Setup, Rerouting. (2^16-2^13+2^8+2^6)

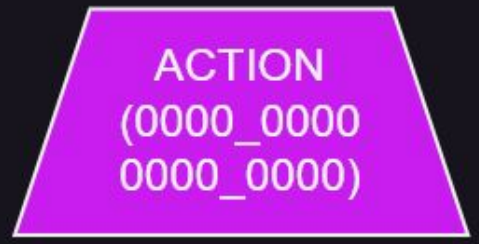
Typical Packet Structure

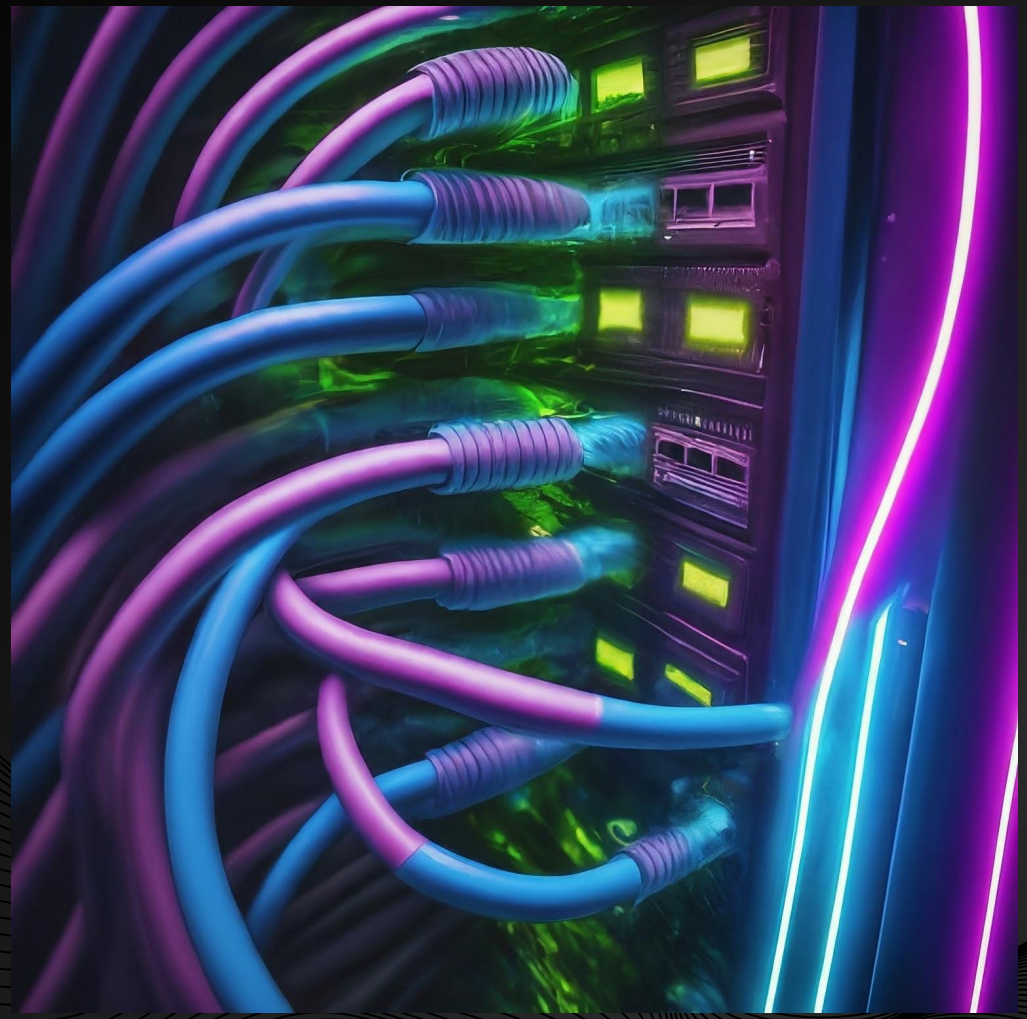
Each packet on port 57344, general data, should contain two ID bytes, followed by two action bytes, and then as many data bytes as needed. It is TCP, so no checksums or intended target info needed.

The ID points to a GPIO pin or sub-object run by a specific module.

The action bytes would differ from module to module with meaning.







Device Chain on Port 57664

A key feature of the network is the possible redundancy, and this would be accomplished in the system administration socket. On initialization, a IP neighbor chain should be determined so that packets sent along the route pass through every device on the network. Ports are client to client, not client to all devices, and this chain remedies this.

Device Chain Process

First

A module is powered up, and determines the closest IP above it, wrapping round 0 if needed. Sending pings to determine active IPs.

Second

It continues to scan for devices between itself and the next known neighbor, to account for changes.

Third

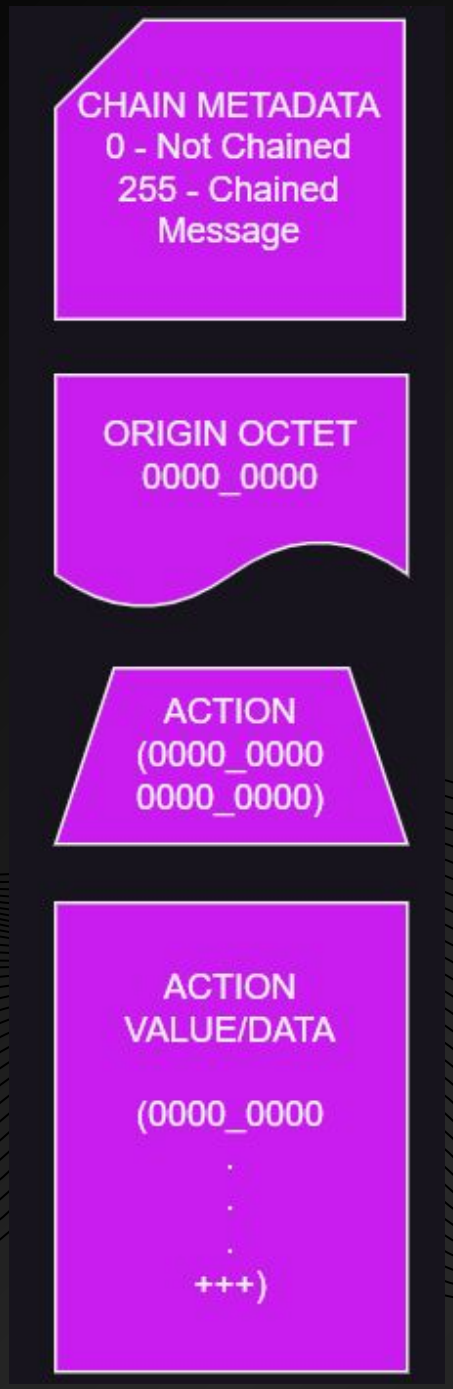
The module sends a “chain” packet, containing special metadata indicating it should be passed down the chain.

Fourth

The neighbor gets the packet and forwards it to its neighbor.

Fifth

Once the packet completes the whole chain, the original module knows the chain is functional and is ready to operate.

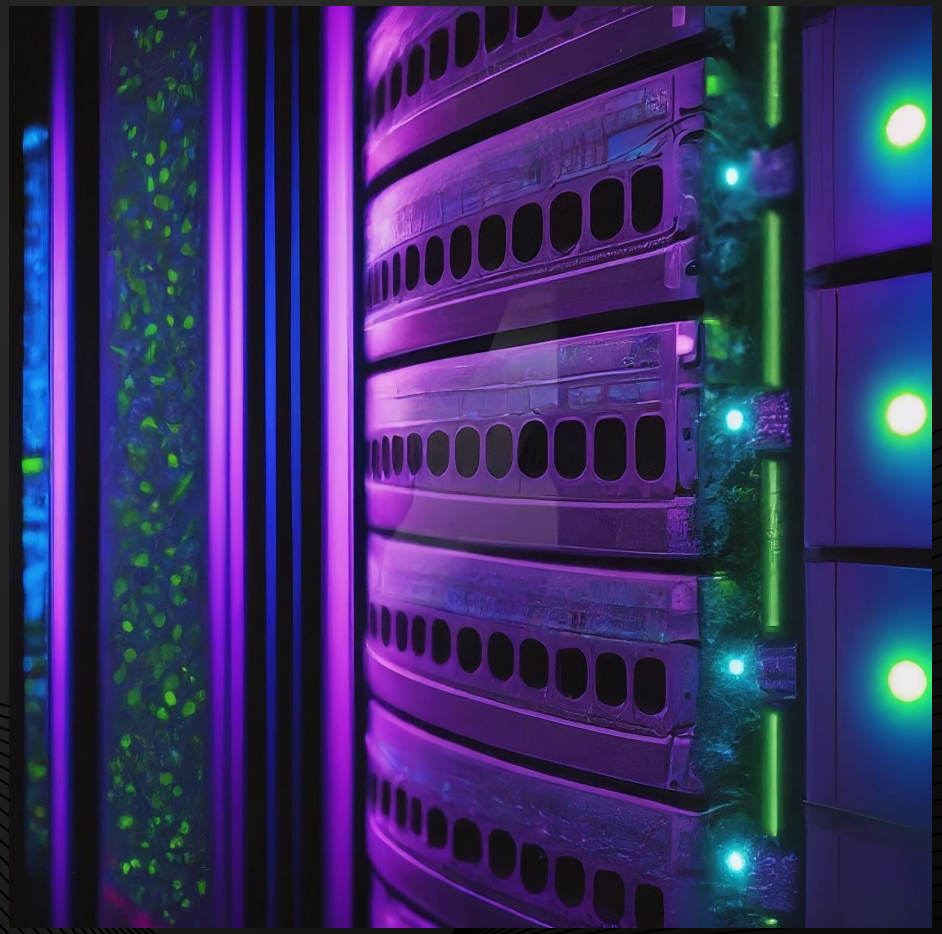


Chain Packet Structure

A packet on the system admin port would look similar to a general data packet, but would include an additional byte at the beginning defining it to be passed down the chain or not.

It would also need a host address at which the packet would stop looping on the chain.

Neighbor Chain Uses



Chain uses include, device status and check-in, interrupt client changes, and anything else where an @all devices or updating something on multiple devices would be useful.

Say the Jetson crashes, and interrupt actions need to go to a backup SBC, a chain message could quickly update the IP of the interrupt service on every module.

A goto sleep power saving command could also be passed through the chain efficiently.

Overall,

Developing this system would be a lot of work the first time for both electrical and software, but the andvangestes of a reusable system would save time year after year as it’s reused.

Any questions on the information covered?

Thank you!

Do you have any questions?