CISC 322

Assignment 3
Apollo Architecture Enhancement
April 18, 2022

Group 37

Karl Dorogy	18kwd1@queensu.ca
Ryan Saweczko	18rjs5@queensu.ca
Tom Lin	18phl@queensu.ca
Briggs Fisher III	18wbfi@queensu.ca
Douglas Chafee	18dsc1@queensu.ca
Itay Ben Shabat	18ijbs@queensu.ca

Abstract

Using the architectural style of implicit invocation (aka Pub-Sub) and the high-level conceptual architecture previously established for the top-level system of the Apollo autonomous driving project, we now propose a new particular feature or enhancement to be implemented within the system labeled the communication module. We decided to propose the addition of a communication module, as well as describe the current state of the system relative to the enhancement with the mapping and discussion of its new interaction and effects within the system.

Additionally we discussed two alternative implementations of the communication module in regards to a client-server and or a peer-to-peer communication network with the potential risks and testing impact that comes with either method being made apparent to the reader. Multiple use cases along with their respective sequence diagrams were produced. Finally, a full SAAM analysis was performed to the Apollo stakeholders on both alternatives to the communication module. Each alternative was successful at providing different NFRs but ultimately the peer-to-peer implementation best fit the desires of the stakeholders.

Introduction & Overview

After learning the ins and outs of Apollo's architecture, both the conceptual and concrete, we are prepared to suggest an addition for the Apollo architecture. After researching their architecture we found an addition that is not implemented and we now know how to properly integrate this addition into the large-scale architecture. We have suggested that addition of a communication module with two different ways to integrate the module into the system; a peer-to-peer integration that communicates the optimal routing to its peers based on its perception of the road including any obstacles or traffic, and a client-server integration that shares data to a server from which the optimal routing is sent back. By communicating with other Apollo devices or an Apollo server, the information known to each car can be fully realized and optimized.

The report is split into five sections. The first section includes first, why we suggested the addition of a communication module as well as an overview for how the feature benefits Apollo users. Second, an introduction to the peer-to-peer and client-server implementations and a demonstration of the integration of these implementations into Apollo's conceptual architecture through figures and explanation. Third, a discussion of the interactions the new module would have with the existing subsystems for both implementations. Finally, a comparison for the differences of the module's interactions between the two implementations. The second section describes the effects the introduction of a communication module will have on the maintainability, evolvability, performance, and testing. Additionally, the second section warns of the risks associated with the addition in regards to the security, maintainability, and performance. The third section provides two different use cases for situations involving our addition alongside their respective sequence diagrams. The fourth section includes a SAAM analysis that lists the major stakeholders for our addition and what nonfunctional requirements they would care about most. Additionally, the fourth section describes how each implementation would satisfy the desired NFR and how the two implementations differ at providing these NFRs. Finally, the fifth

section includes a data dictionary with definitions for technical terms used throughout the report, and naming conventions in a list of acronyms with their corresponding unabbreviated forms.

From researching how our addition would fit inside the conceptual architecture of Apollo alongside how the existing modules would interact with both implementations to researching what NFRs our stakeholders would desire and how our communication module's implementations would provide them; we have decided that a peer-to-peer implementation of a communication module would best fit the NFRs for the majority of the stakeholders and its only two downsides of security and performance can be handled to mitigate their negative effects against their NFRs, resulting in a positive addition overall. Developing and reporting our communication module has only reassured us that the benefits our addition would bring are well worth the effort to implement, and will fit well within Apollo's architecture.

1. Proposed Enhancement

The current Apollo system is designed so each car runs as an independent entity with no communication or awareness of other Apollo vehicles around it. Operating as an independent system, Apollo vehicles rely on their own perception abilities through camera, Lidar, and RADAR to safely navigate from point A to B. Objects perceived here are used in the prediction module to predict the trajectory and location of the obstacles. With this information about obstacles from the prediction, the planning module runs the decision process to determine the best action when approaching intersections, roundabouts, or any other obstacle. Planning relies on prediction to determine the trajectory and location of obstacles to inform the controller of any changes to direction or speed required to safely navigate the route. Every single car that uses Apollo has these systems running independently, so each car can determine the best actions and route to take on its own.

A possible enhancement to the system is adding a method of communication between vehicles, taking them from independent entities to a network of autonomous cars. This addition would add a level of optimization to routing and reduce travel times in urban environments where congestion is a common problem on roads.

The immediate effects this would have on the system is implementing a new driver for wifi capabilities, and a new communications module. This line of communication would impact prediction, planning, and routing, using information other cars may send to improve these elements. However, the Apollo system would still be capable of routing and driving as an independent system without any other cars nearby for this module to take effect.

1.1. Conceptual Architectures of Proposed Enhancement

We are proposing two different ways to implement a communication module that will allow multiple Apollo vehicles to communicate to each other to better plan their routes. The first method will be a peer-to-peer communication between all Apollo cars in close proximity (Figure 1). This will act like a hive mind, where all of the cars are able to send information to each other for planning and routing purposes. For example, if an accident happens on a highway, an Apollo car near the accident will be able to broadcast this event back to other cars farther back, giving them a chance to exit the highway earlier than they would have to avoid traffic from the accident.

This information would be propagated in a wave from the area of the accident as more Apollo cars rebroadcast this information to other cars farther away.

The second method the communication module can be implemented in is in a client-server style (Figure 2). With this method, Apollo can create a single central server that all clients using Apollo can connect to. Each car can then send its location, as well as its current route and detected obstacles to the server, which will collect the information and inform cars of a more optimal route using all the information it receives. Given the accident example from above, the car using Apollo would send information about the accident to the server, which can then recompute better routes for all other vehicles in the area to avoid the accident, then inform each car of this new information.

With this enhancement, the autonomous cars will still need to be fully functional without the communication module. This module will only supply suggestions and optimizations to the routes cars can take, but each car still needs to be able to determine a route and plan on its own. There won't be a guarantee that the server or peers will be there for communication 100% of the time.

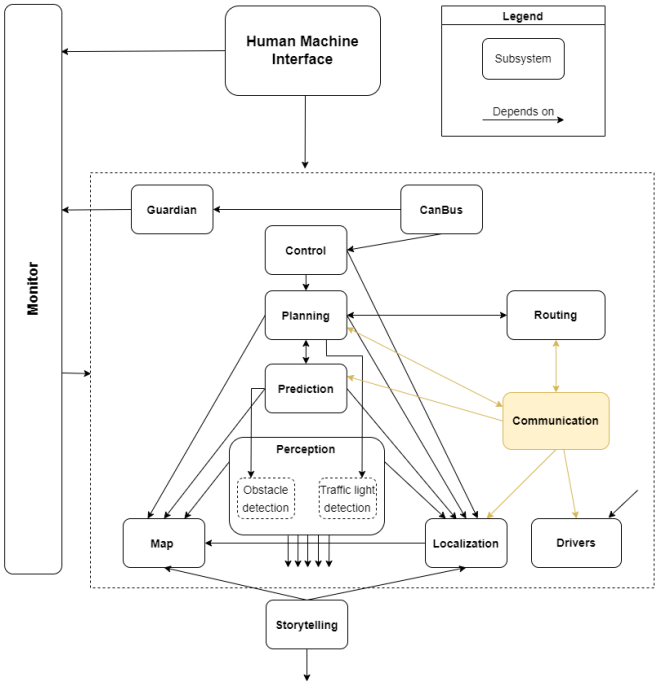

Figure 1: Enhanced Conceptual Architecture of Apollo with Peer to Peer Communication.

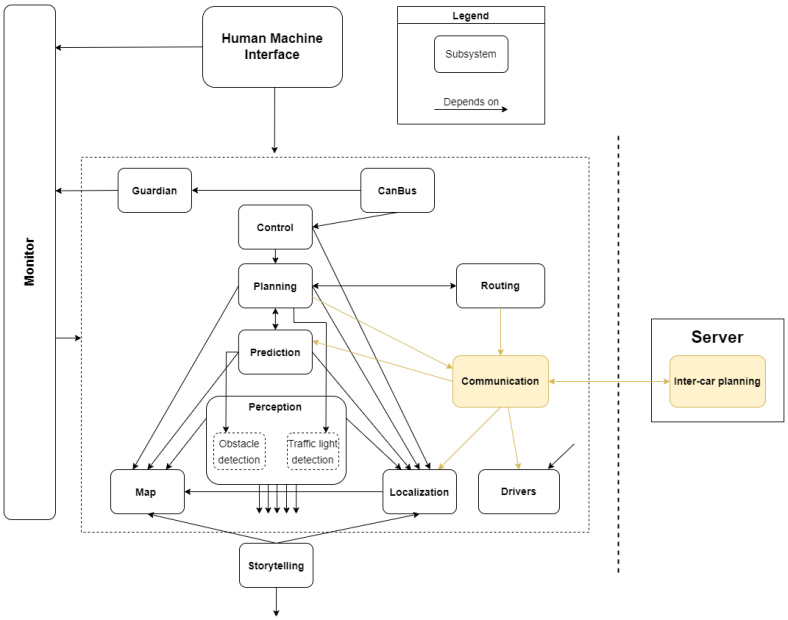

Figure 2: Enhanced Conceptual Architecture of Apollo with Client Server Communication.

1.2. New Interactions within the System

From observing the modules within the software architecture and comparing it with our proposed enhancements of the product, we can conclude the modules that will be impacted upon implementing our changes will be drivers, planning, prediction, localization, and routing.

The prediction module is responsible for determining what obstacle was perceived in the perception module. It would predict what the obstacle data perceived was as well as its velocity,

such as moving in one lane, across lanes, or not moving. By adding our enhancement, the communications module would use the prediction module solely for its predictor submodule's output. The predictor submodule is the specific submodule used to determine the type and velocity of nearby obstacles. The communications submodule will use this output and broadcast a subset of the data it deems "important" to surrounding cars. Anyone nearby with an Apollo car will be notified of such obstacles in advance and can then decide their next course of action, whether that is to continue on-route or to plan a reroute around the oncoming obstacle. Things deemed important may vary, but would include obstacles likely to cause long lasting obstructions in the road, such as a crashed vehicle. The specific directory that will be impacted will be prediction/predictor, with the outputted cyber channel from prediction_component.cc, specifically PredictionComponent::PredictionEndToEndProc being a channel the communications module will listen to.

The Planning module is responsible for utilizing information received from the Prediction module. It would coordinate the next moves the autonomous vehicle would take according to the oncoming obstacle data. With our enhancement, the planning module would also account for obstacle data incoming from other Apollo vehicles. It's possible vehicles in slightly different positions will be able to see different cars, so this method can improve accuracy for obstacle avoidance. In planning_component.cc, specifically PlanningComponent::Init, a new node to CreateReader for a CommunicationMessage can be added as a cyber channel from the new module. This new node will be used in the planning/planner and planning/tasks directories, which deal with the 'what' part of obstacle avoidance. By using the new channel's information inside these directories, the obstacles detected by other vehicles and sent through the communication module can be used to help plan this vehicle path.

The Localization module provides localization services, such as Real Time Kinetic information and information about the car's current GPS location. The communication module will take the car's location from a LocalizationEstimate (defined in modules/localization/proto/localization.proto) and broadcast this to other Apollo cars in the area. This location is an estimate of the car's actual location determined by multi-sensor fusion in the localization/msf module. Multisensor fusion incorporates data from GPS, Inertial Measurement Unit, and LiDAR to get as accurate a location for this car as possible. It isn't likely this directory (or any other directory, like localization/ndt or localization/rtk which also provide estimates for the car's location) will need to be modified for this addition, rather the communication module will need to listen to the cyber channels these estimates are already broadcasted on. No information from the communication module will need to go back into localization, so changes to it will be minimal if required at all.

The Drivers module contains the required drivers for peripherals the autonomous cars require, such as microphones, lidar, camera, and radar, and streams the output on cyber channels for other modules to use. By adding our enhancement, we will be adding a new driver component, for Wifi. This new driver will allow the communications module to communicate with other Apollo vehicles via WiFi (given the vehicle has the required hardware). The only change required to the directory structure of the drivers is a new folder to contain the Wifi driver.

The Routing module creates the route or path the vehicle is going to follow to get to its destination. It plans the route to destination at the start, and updates the route as needed when

large obstructions are in the way forcing changes to the route needed. With the communication module two different improvements will be possible: optimal routing to avoid congestion and early avoidance. The routing module's routing plan can be sent through the communication module to other nearby cars so they can plan around each other. Furthermore, information from the communication module such as predicted obstacles (ie other car's found an accident on the road and broadcasted the information and location) can be used in routing to preemptively plan a route that will avoid the location of the accident. The likely changes to the routing module for this would be routing.cc and routing_component.cc, as well as the routing/core directory.

1.3. Client-Server specific changes

The only change specific to the client-server implementation style would be a connection between the communication module and a newly built server. The server would need to be implemented as a part of this enhancement on top of the communication module itself. The communication between the car (client) and server would be the main load on the car itself, and the server would do all of the computation to determine routing and plans for cars in an area, spreading that information to cars the plan affects. The communication module then informs each client about this plan, sending it to the routing and planning modules to be used.

1.4 Peer-to-peer specific changes

There are two changes peer-to-peer has that the client-server implementation doesn't. They are dependencies for the communication module directly depending on both routing and planning. Since with peer-to-peer there is no overarching server that can see all of the car's locations, predictions and intentions, each car needs to stream more information into the communication module for other cars to find. This additional information includes the routing information and the car's current plan, since the additional information is needed to allow the cars to optimize their routes despite having less information due to the limitations of peer-to-peer.

2. Consequences of the Enhancement

2.1. Effects

The system should remain as easy to maintain as before. The cars should be able to operate properly even if there is a failure in the communication module. This must be the case since a car using the peer-to-peer implementation that is alone on the road will be required to function without the communication module being able to provide any help. Similarly for the client-server implementation, a car should not fail if the server crashes or the car goes out of range of any wifi signal. Maintenance isn't going to be a critical problem within the implementation of the system, where any crash could be catastrophic, any failure in the communication module will not result in critical failure. The need for maintenance in a client-server implementation is slightly more important since if the server is not maintained and fails then the communication modules for all cars connected to the server will fail; this implementation results in a single point-of-failure for this module in all cars if something happens to the server. On the other hand, maintaining the peer-to-peer implementation is done on

a car to car basis where any car's failure will require their own maintenance, however their requirement for maintenance will not directly affect any of the other cars.

The evolvability of the system will remain relatively simplistic for most ways this change could be implemented. Since the communication module will be distinct from the rest of the driving functionality, testing the change and adding new features will not affect the risk factor of using newer technology in a self-driving car. In either a client-server or peer-to-peer implementation the client (or at least two peers) need to be updated or properly versioned to allow the new capabilities added in an update to be utilized by the cars. However, with proper versioning and backwards compatibility updates can be handled properly without conflicts.

The performance of the client-server implementation will be bottlenecked by the performance of the network, and the performance for the peer-to-peer implementation will depend on the network connection between each peer and their respective computing power. There will be more options for connections on the peer-to-peer implementation in case of network issues or failing, however the connections will often be less stable than a server's connection. The server's performance will be more consistent as there are no guaranteed responses from peers who could be down, delayed, or even hostile. However, since the communication module is only for optimization in routes and planning, performance problems in the network are ok and won't affect the car's ability to drive.

2.2. Testing Impact

Some tests that can be performed irregardless of what method is used for implementation. Testing the driver for wifi and the cyber channels for the new communication module to ensure they function, with unit tests to ensure the functionality remains into the future. Having unit tests with different inputs to ensure in a minimal environment the expected behaviour occurs would be ideal. Testing the entire system may vary depending on if a peer-to-peer or client-server implementation is chosen.

The Apollo software has the capability to run test scenarios to see how a car would react in different situations already. In a peer-to-peer setting, testing the new capability in the overall architecture and environment of the car could be done using that software. By running a simulation of two or more cars on a highway with the communication module active, a peer-to-peer style implementation can be tested by adding an accident ahead of the first car, with a large distance between each of the Apollo controlled cars. Then, when the first car relays information to the peers in the simulation we can see if the communication module is functioning by the other car's reaction, turning off the highway, or checking the other cars routing through the HDmap interface in the simulation.

Testing the client-server interaction in the full environment might be a little harder than peer-to-peer due to the requirement of also hosting a server. A test suite with a way to host a local version of the communication server so a user can run a simulation and their own server would likely be required. With this setup, a similar situation to the peer-to-peer could be run, using their own local server to not send simulation messages to the real one for active cars. Outside of the simulation testing, we can test things like response times from the server and if the communication module slows down the total speed of the system and by how much. This is done to make sure that this extra module won't negatively affect the system.

2.3. Potential Risks

With the newly proposed communication enhancement of the Apollo system comes a few potential risks. Firstly, regarding the client-server method of implementation, the inclusion of a single central server in which all Apollo clients can connect to and send their respective information to, creates a significant single point of failure/risk in regards to availability within the system. Additionally with this availability risk, a client server connection also makes the quality/performance of the Apollo system's routing and planning modules semi-dependent on the overall network's performance at times, potentially risking the response time performance of vital information to the system. Even though the communication module only supplies suggestions and optimizations to the currently active Apollo cars and their routing/planning modules, a significant lack or delay of information caused by a down server or high traffic network will still severely impact the overall quality of routing between Apollo cars as a whole in terms of car congestion, traffic, and accident avoidance. Furthermore, server performance in regards to the throughput of information is also a major potential risk in which Apollo's central server should be able to handle and compute all incoming and outgoing information with minimum delay even during peak hours such as national holidays and or around 5:00pm on a week day when most individuals start their commutes. Lastly, there is a potential scalability risk as the number of operational Apollo cars grows eventually and how many clients can be connected to the single central server or network before performance response time and throughput cannot be upheld anymore.

Generally, across both the client-server and the peer to peer method of implementation, a major potential risk that both implementation methods hold is security. Security risks, whether they be the sending of false/malicious routing information/messages to other cars, altering messages in transit to/from cars, and or the interception of private information over a client server or car to car (peer-to-peer) communication are all potential risks introduced to the architecture with the addition of the communication module enhancement. The potential authentication, authorization, encryption, and integrity risks that are introduced would have to be both tested and validated so that malicious actors can't directly interact and negatively impact currently active Apollo cars. Regarding more specific security cases, in the case for a client-server communication, DDoS attacks are especially apparent in creating an availability risk, where hackers could execute distributed denial of service attacks blinding the vehicle from information to the outside world[1]. In the case for a peer-to-peer communication, the sending of malicious information or manipulation of information to the autonomous vehicle's communication module allows attackers to "hijack" communication channels between cars and manipulate obstacle detections, in which they can then wrongly propagate false information through the network and or block an accident event's information from being distributed to other cars immediately entirely for example.

3. Use Cases

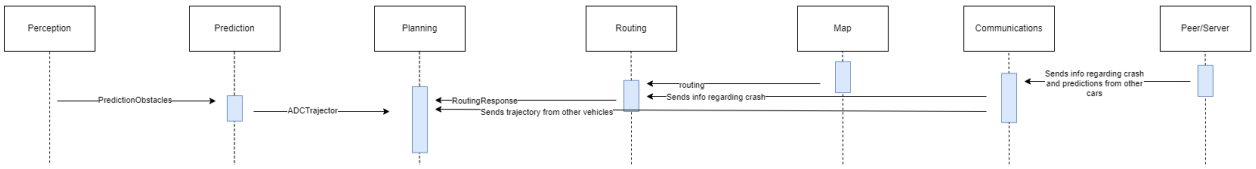

Figure 3: A new crash has been reported and the routing adjusts based on this

For this use case, a car sees that there is a crash ahead of it, or the car is part of the crash, and it sends this information to the communications module which then sends it to the server or to surrounding cars, depending on the implementation style. When the communications modules of the cars receive this information, they send this information to the routing modules so that they can use this new crash when calculating the new route. The communications modules also send trajectory info from the server to the planning modules.

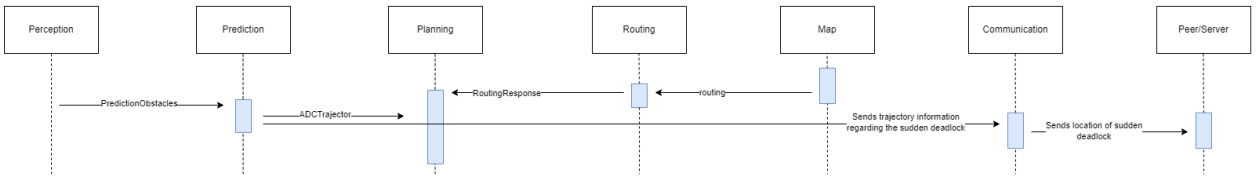

Figure 4: A car comes to a sudden deadlock and sends info to server

For this use case, a car comes across an unforeseen deadlock on the highway. The prediction module detects the deadlock and then sends this info to the communications module which then sends it to the server or to other cars around it, depending on the implementation style. This way, when cars behind this car get this info, they can either reroute and go through a different way if they can, but if not, they will know to slow down when coming getting close to the deadlock, as it won't be a surprise to them like it was for the original car.

4. SAAM Analysis

There are a few major stakeholders of this proposed enhancement. The major stakeholders include the developers of Apollo who would be implementing this feature, the owners/creators of Apollo who need to front any costs and ensure it works properly, and the users of Apollo in areas where multiple people use Apollo are, who will be the main group affected by this change.

Some of the important nonfunctional requirements for the developers of Apollo would be maintainability. The developers would want tests that can be run easily (easy to set up the environment required to test the communication) and quickly (within a dozen minutes to an hour in development) to ensure the new module continues to function properly. Another NFR for the developers is modifiability. The new module should be implemented with interactions only to the

other modules it is specifically required to interact with, minimizing the dependencies to allow for easier modification in the future.

The users of Apollo would have two major NFRs: security and performance. For security, the users would want a large amount of security on the communication network, to allow only "valid" Apollo cars to send messages, and filter out messages by malicious entities. With performance, users of Apollo would want the communication to take as little processing power from their car's computer as possible. Since this feature is an enhancement that won't be critical to the capability of the cars to drive, mostly acting as an optimization in routing/planning, it shouldn't consume too many resources that could otherwise be used to improve accuracy in avoiding obstacles or driving properly.

The final stakeholder, the owners of Apollo, would care about the scalability of the communication module, how many cars (say 1 thousand) can be connected to the network before the response time starts to increase too much. The owners would also care about security, similar to the users. If this system gets hacked and malicious instructions are sent to cars due to that, it would cripple the company due to lack of trust from their users.

Two potential ways to realize the implementation of a communication module would be to add it in a peer-to-peer style, where the cars communicate directly to other cars, or in a client-server style, where the cars send their information to a single server that then sends instructions to all cars.

4.1. Peer to Peer implementation

With the peer-to-peer based implementation, the NFRs that will almost definitely be positively impacted are scalability, maintainability, and modifiability. Since the cars will be talking to other cars in the immediate vicinity only, the model will scale very well if Apollo is spread over the world. The only limit to scalability immediately is how many Apollo cars exist within the range of the wifi signal. There could be millions of cars worldwide without affecting the performance if they are spread through sites across the world. Maintainability would be relatively easy since this module shouldn't have too many interactions with the other modules, and is more an information-relay module than anything else. Since this module is intended to relay information from one car to another, there won't be much, if any AI to test, and testing to ensure the communication works should only require two instances of the Apollo software. Similarly for modifiability, since the module is relatively independent from most of the other systems, being the cars should remain fully functional without the communication module, there will be relatively low risk and challenge to modifications to it, since any problems in the modification won't result in the cars driving poorly.

The performance NFR would be slightly more challenging for this implementation. With only a few other Apollo cars near one user's car, the overhead for the communication will be low. However, this overhead would greatly increase as more cars are nearby, and multiple cars all reporting the same events might cause feedback loops and more performance loss. This is different from the scalability since the cars would be able to handle many other Apollo cars nearby, it's just the CPU and bandwidth usage would grow above a reasonable bound, requiring users to get better CPUs before any scalability limit would be reached. This problem can be mitigated to a reasonable degree by reducing the events cars will broadcast to each other if there

are more cars nearby, limiting broadcasts to only be large obstructions on the road, or other serious impediments that would strongly affect other cars in the area.

The final NFR, security, would be the hardest impacted with a peer-to-peer implementation. Since Apollo is open source, anyone, including malicious actors, are able to download and use the software. In a peer-to-peer system it would be incredibly hard to police all the cars in the network to stop such actors. We are unsure of what cryptography might exist that would allow cars to be trusted to be sending only valid, accurate information and not faking accidents or other road conditions.

4.2. Client-Server implementation

In a client-server implementation, the NFRs that would be easily met are maintainability and performance. Similar to with peer-to-peer, the added module should be mostly distinct from the rest of the autonomous driving capability. Thus, testing this new component can be done in a very small environment and focused only on ensuring connectivity to the server to send and receive data. The server would also need its own testing components, which wouldn't be too bad since it is also a distinct entity that can be tested with unit tests. Performance overhead in each car from this type of communication would also be very low, since each car only needs to talk to a single server, which does the processing. Since any processing the server needs to do isn't a factor in the performance metrics of this NFR, since client's don't care much if information about obstacles/problems farther down the road is a couple seconds delayed, the large load on the server that may slow it down isn't a problem.

Scalability, modifiability, and security are all impacted a little harder by this change, and would be a bit harder to reach. Scalability is a problem because instead of only nearby cars communicating in peer-to-peer, now all cars are connecting to a server, increasing the load more. This can be mitigated by expanding out, adding new servers to handle cars in specific cities / countries so the server never gets overloaded. Since the benefit of the communication module is limited to help cars navigate based on other local information, this will be an effective way to scale. Modifiability is also harder because of the increased complexity having a central server that cars need to connect to will add. Instead of each car being its own ecosystem, which remains mostly true for peer-to-peer, now a setup will require a separate entity to communicate to. Updating the server will need a complete test setup for the server, and more work to integrate the change seamlessly to cars that might be a slightly different version. Finally, security is hard to implement because the Apollo software is open source. However, since there would need to be a central server owned and controlled by Apollo, they would be able to create tokens to give to trusted cars as one solution, or at least the central server would be better equipped to detect malicious information given by one car by seeing if other cars in the same area also report it. This allows the central server some power to determine if a specific car is malicious.

4.3. Comparison of the two methods.

Each implementation impacts the NFRs in different ways. For the developers, the clear preferred option would be a peer-to-peer implementation. Both implementations would have roughly the same level of maintainability, but peer-to-peer would be easier for them to create and modify. The difficulty with the client-server here is that the entire server part of it would need to

be built from scratch since no such server currently exists. This adds some complexity to maintaining the system, and a lot more complexity to modifying it once the servers exist. From the user's perspective, the client-server implementation is preferable. Having a server to connect to alleviates both of the major concerns of the client: security and performance. Having a separate server to communicate with would simplify the processing required in each car to handle a lot of communication from all other cars, and also provide a way to add security against malicious actors in the system. Finally, the owners of Apollo won't have too strong an opinion about which implementation to choose. The peer-to-peer implementation would be easily scalable, since it mostly scales by itself without any necessary changes, while the client-server has better security capacity for them. However, another slight requirement the owners might have that could push them to peer-to-peer is the simpler infrastructure (or lack thereof) required for peer-to-peer. There is no need for a server they need to maintain to exist in that situation.

Given this analysis of the two proposed implementations for a communication module, we believe the best one would be a peer-to-peer style implementation. The only major drawback of this implementation is the security of the system. Creating security in the peer-to-peer system would have to be given a high priority, and could be achievable through consensus of multiple messages, where if one car alerts with a message, receiving cars won't necessarily follow it unless other cars also in the area report seeing it. The cars can screen messages and decide which to follow or not. The messages don't directly control the car, thus malicious actors in this sense could be mitigated slightly. Furthermore, the performance penalty that may occur if there are many Apollo cars in the same area can be mitigated through careful control of what messages are sent. If there's a high density of Apollo cars their peer-to-peer messaging scheme can change to only report large amounts of traffic or accidents, giving less of the finer details about the state of the world around them to other cars. This would still allow for improvements in the car's planning capabilities to avoid high-traffic areas, while not hurting performance as much in areas with a high density of Apollo cars.

Thus both of the major downsides to a peer-to-peer style can be handled to some degree to mitigate the negative effect against their NFRs, so the peer-to-peer style is the better implementation.

5. Data Dictionary

Client-Server: The architectural style where distributed data and processing is done across a range of components remotely over a network. Servers are stand-alone components that provide specific services while Clients call on those services.

Peer-to-Peer:The architectural style where tasks/workloads are partitioned between peers without centralized control. Peers are stand-alone components that provide specific services to each other over some kind of direct network connection.

DDos: Distributed denial of service attacks involves multiple connected online devices, collectively known as a botnet, which are used to overwhelm a target with fake traffic limiting availability to the target.

5.1 Naming conventions

LiDAR: Light Detection and Ranging

RADAR: Radio Detection And Ranging

HMI: Human-Machine Interface

RTK: Real Time Kinematic

IMU: Inertial Measurement Unit

GPS: Global Positioning System

NFR: Non-functional requirement

DDos: Distributed denial of service attacks

References

1. https://www.cpomagazine.com/cyber-security/eu-agency-for-cybersecurity-says-autonomous-vehicles-highly-vulnerable-to-various-cybersecurity-challenges/