CISC 322
Assignment 3

Enhancement Proposal for Apollo's Open Software Platform

Friday, April 8, 2022

Apollo-gies in Advance

Cameron Beaulieu	19cgb@queensu.ca
Truman Be	18tb18@queensu.ca
Isabella Enriquez	18ipe@queensu.ca
Josh Graham	18jg48@queensu.ca
Jessica Li	19jal@queensu.ca
Marc Kevin Quijalvo	18mkq@queensu.ca

Table of Contents

Abstract

This report proposes a new feature for the open-source autonomous driving platform, Apollo. Entitled "Cross-Communication," this feature is an entirely new module which enables communication between Apollo vehicles for sharing information such as the relative map and object tracking list to increase the perceiving scope of each Apollo vehicle. Additionally, the feature allows user vehicles to warn neighbouring vehicles when it is experiencing failure and may become a hazard. Through these two generalized use cases, Cross-Communication helps Apollo vehicles more accurately predict future obstacles and enables further optimized routing.

Following a more detailed overview of Cross-Communication, the report looks at two different ways this new feature could be implemented, including a SEI SAAM architectural analysis for both approaches. The impact of the feature on the existing architecture and subsystems at both the conceptual and concrete levels—including the risks the accompanying changes in the architecture may introduce, and plans to test the impact—are also explored. To further illustrate the value of Cross-Communication, the report investigates two key use cases: (i) when a nearby vehicle perceives a major obstacle, and (ii) when the user vehicle is experiencing module failure.

Finally, the report concludes with a summary of Cross-Communication and a description of the lessons our team learned in devising a new feature for Apollo.

Introduction

As the technological field continues to rapidly evolve, autonomous vehicles are at the forefront of innovation, and Baidu's Apollo is nothing short of extraordinary. The project was launched in 2017 and consists of an open-source autonomous driving platform that enables developers and stakeholders to develop, test, and deploy autonomous vehicles. Apollo consists of an extensive list of features and modules, is partnered with various companies around the world, and boasts more than 200 individual contributors (*Apollo's GitHub Page*).

In our team's previous reports, we explored the conceptual and concrete architecture of the Apollo system. Now, we are introducing a new subsystem: the Cross-Communication module. In a 2016 report, studies showed that Toronto drivers lose 20 minutes for every 30 minutes of an evening commute when stuck in congestion (Montgomery, 2020). Traffic, car accidents, and construction are all unexpected obstacles drivers can come across during their daily commute that can significantly slow them down. Our Cross-Communication module implements the ability for vehicles to relay this information to each other to plan for more optimal routes to save the driver's time. This module can also greatly help with accident prevention by alerting nearby vehicles so they can act accordingly and adjust their navigational path.

For this report, we will first be providing an overview of what the purpose of the Cross-Communication module is and how it operates. We also dive into two potential implementations of the module, including a Client-Server style and a Peer-to-Peer style. We then build a conceptual architecture for each approach and observe the new dependencies created by the addition of the new module within the chosen implementation. With our SEI SAAM analysis, we weigh the benefits and drawbacks of each approach with regards to key stakeholders and their corresponding non-functional requirements. After deciding on an implementation, we explore the impact the addition

of this module has on Apollo's concrete architecture and other subsystems as a result of the introduction of new files and dependencies. The addition of a new feature also means that new risks are introduced, so we further explore the limitations of our Cross-Communication module, how the module should be tested, and ways in which the system could become compromised. After analyzing the details and specifications of the new feature, we describe two use cases in which our module can prove to be useful through the use of sequence diagrams. This includes a crash detection and diversion scenario, as well as a car failure alert. We conclude the report by summarizing our learnings from the research and ideation of the Cross-Communication module.

Overview of the New Feature

As autonomous vehicles become more of a commonality on the road, this will likely result in the increase in Apollo-controlled vehicles out in society. Our team wishes to leverage this increase in hopes of improving the driving experience through improved routing and increased safety, thus we present the Cross-Communication feature. The Cross-Communication feature allows for communication between Apollo vehicles, including the sharing of object tracking lists, relative maps, and the statuses of each Apollo vehicle. Similar to apps such as Waze, the sharing of the relative map and object tracking list will allow for the vehicle to detect possible cautionary obstacles further down the current route (outside of the car's perception range), such as a car crash or traffic jam, and thus will allow the vehicle to adjust the planned route to avoid this area. Provided that Apollo vehicles eventually make up a large enough proportion of the vehicles driving on the road, this feature would also allow for the optimization of the routes for all cars, based on the number of cars currently driving each route and each route's optimal capacity. As well, in the event that an Apollo vehicle has an issue with a module or some piece of hardware, requiring the Guardian module system to immediately stop the car, the Cross-Communication module will allow the car to make other vehicles aware so that the Prediction module can identify the vehicle as a "cautionary object" and allow the Planning module to handle it accordingly.

Approach 1

Our first approach for implementing the Cross-Communication feature is to introduce it using a Client-Server style (see Figure 1). Vehicles would solely interact with the Cross-Communication Server. When they send data to the server, the server will, in turn, send the data over to the Cross-Communication module which will relay the appropriate information to its subscribed submodules. It will send a response back to the server, which will then forward the information to the vehicle that sent the request. With this approach, vehicles will be able to access data from the server regardless of their geographical location and will not be limited to any range. This implementation also allows for more security since all vehicles will be interacting with the server instead of each other. The disadvantages of this implementation, however, lie within the module's dependency of the server. If the server happens to fail, the entire network will go down and all vehicles will lose access to the module. Additionally, the server will be handling requests from all the vehicles using the module, meaning that this could drastically slow down the server and cause severe latency, delay, and poor performance. This could be due to the location of the car relative to the server. If there is only one central server,

performance will decrease the further the vehicle travels from it. The only solution to this problem would be to introduce multiple servers but this would also be very resource-intensive.

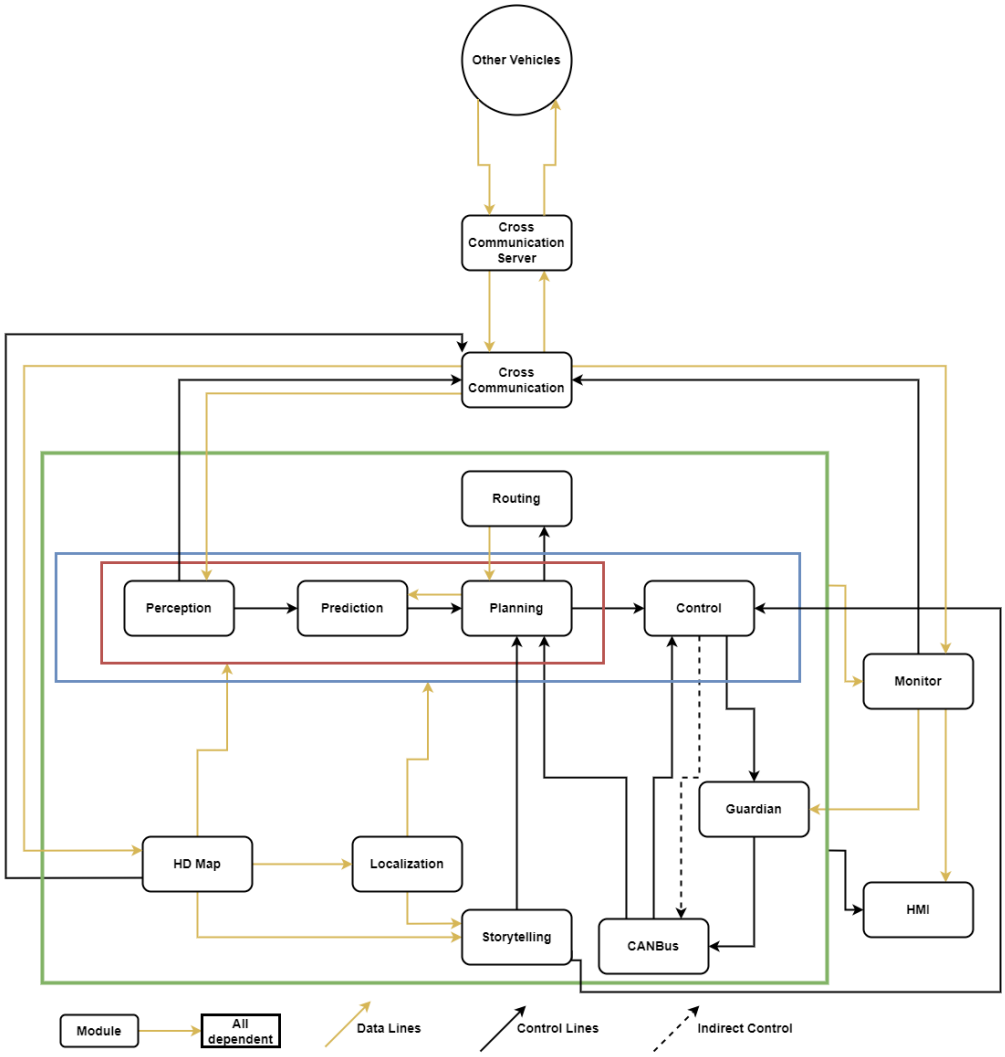

Figure 1. Implementation 1 of Cross-Communication Feature

Approach 2

The second approach for implementing the Cross-Communication feature is through the introduction of a new module, the Cross-Communication module, and the addition of a Peer-to-Peer aspect to the system's architecture, thus creating an avenue for the communication between Apollo vehicles (see Figure 2). The Peer-to-Peer aspect of the system enables the Cross-Communication module to communicate with Apollo vehicles within its communication range, but also access data from vehicles outside of its current range through drawing upon other cars (peers) within the network. This could be highly beneficial in the case that a crash has occurred 50km away from the car's current location, as the vehicles near the crash would be able to broadcast the crash and associated traffic delay to each vehicle through the network of peers, which would act as the peers within the network. This would allow Apollo vehicles to change route,

thus speeding up the time it takes for the car to reach its destination. The advantage to this implementation is that there is no single point of failure. If one vehicle's Cross-Communication module happens to fail, all other vehicles in the system can continue operating as normal. This implementation also makes it very easy to add and remove cars to and from the network. The downside of this system, however, is that it can compromise security since vehicles will be continuously communicating with each other. Additionally, vehicles can become "disconnected" from the network if there are no other cars within the radius, rendering the Cross-Communication module useless.

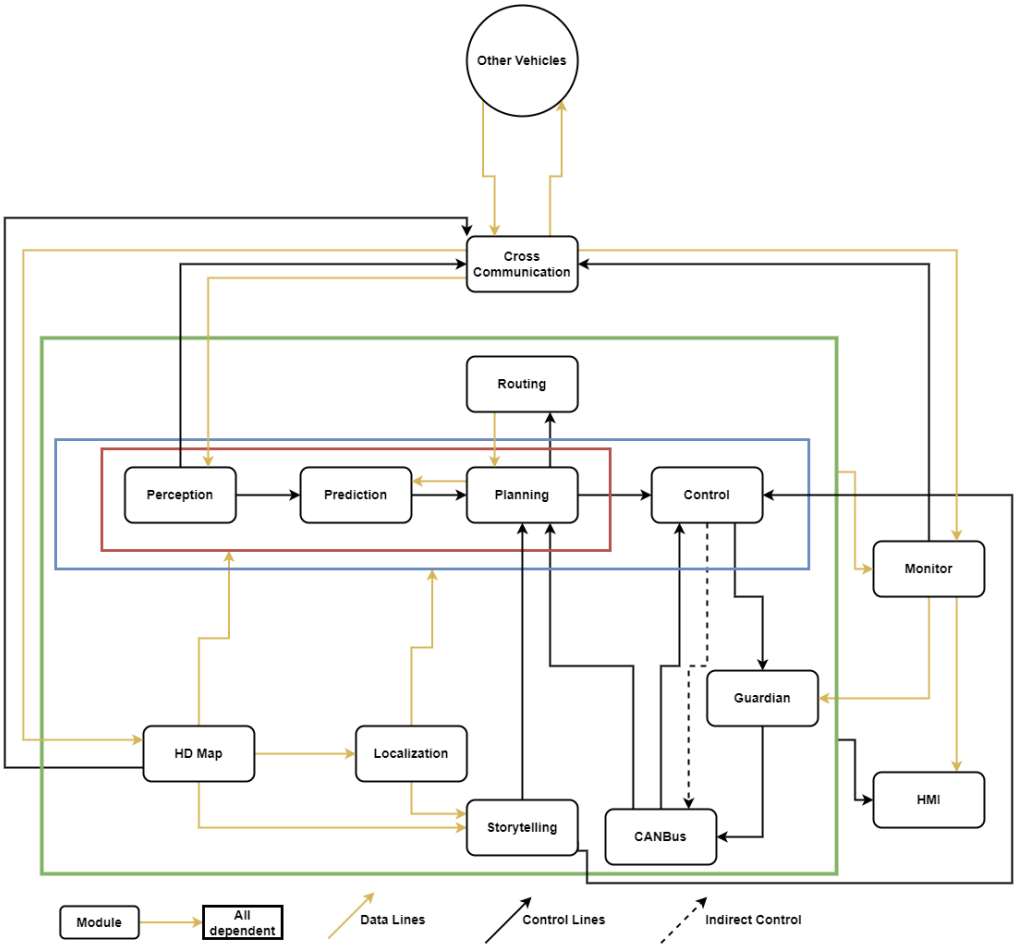

Figure 2. Implementation 2 of Cross-Communication Feature

SAAM Analysis

Stakeholders

There are three primary groups of stakeholders that the Cross-Communication enhancement would affect: users, developers, and investors. Users include anyone that is running, or intends to run, the Apollo software with their car. Developers can be split into two groups: open-source developers, and the Apollo development team. Investors involve anyone that has invested money to help fund the Apollo project.

Users

The most important NFRs for users regarding the Cross-Communication

enhancement are: usability, safety, reliability, and privacy. No matter which approach is used, usability should be almost the same because, regardless of the architecture, the functionality should be very similar. Safety becomes a concern for the user if there is a weak connection to other users causing slow retrieval of tracking list information. This can occur with both approaches. For Client-Server, if the user is in an area with bad reception, transferring data from the server may take longer than usual. Data transfer speeds may also be made slower because with one central server, there may be a large number of requests that need to be processed all together during busy times. For Peer-to-Peer, issues can occur if two cars are trying to communicate from the maximum possible distance range. The connection between the cars will be weaker with a large distance between them, resulting in slower data transfer. There are strengths and weaknesses for reliability with both approaches. With Client-Server, as long as the user is in a location with strong internet connection, the system should be very reliable for receiving information from other users no matter their proximity. However, if there is a server outage, that issue will be widespread for all users. In contrast, module failure in a Peer-To-Peer network will only affect the user whose module failed and any cars in close proximity (assuming that their car is not a single point of failure within the network of cars). In terms of privacy, the Peer-to-Peer approach is continuously sending data between cars which may make users feel their privacy is being violated. However, if any data that could be considered vulnerable is intercepted, it would only affect the associated cars. With the Client-Server approach, data is not being transferred directly between cars, however, if the server was to be compromised, it could affect the entire user base.

Developers

The most important NFR for developers is the maintainability of the system. This requirement should not be impacted significantly differently for either approach. Client-Server style architecture would introduce an extra module for the server, which could make the system more difficult to maintain. However, with either approach, the maintainability of the system is still heavily dependent on how well it is implemented by the developers. If best practices are followed and proper tests are written for the code, either approach should be equally maintainable.

Investors

The biggest concerns that the Cross-Communication enhancement introduces for investors are legal responsibility and security. Investors will need to be confident that the system will function just as well, if not better than it did without the enhancement. If the enhancement has a risk of increasing the rate of component failure, that could result in more car accidents, which will cause lawsuits. With the Client-Server approach, the risk of the system being hacked is a significant legal concern because it can affect the entire user base. Peer-to-Peer also has security concerns due to potential malicious peers, however, hacking would not have a widespread impact with this approach, making it a much smaller legal issue.

Architecture Choice

After analyzing the different ways the two approaches impact the most important

NFRs of the stakeholders, we have decided the Cross-Communication enhancement should be implemented with a Peer-to-Peer architecture. A Client-Server architecture presents too many risks without justified cause by having a single point of failure in the event of something going wrong. Although there are similar downfalls that can occur with both architectures, with Peer-to-Peer, those downfalls are always localized to a small set of users instead of affecting the overall user base. This means that Peer-to-Peer will be better for keeping users safe and secure while being able to trust the reliability of the system. For developers, the choice between architectures should not have a large impact because the maintainability of the system depends on how they have implemented it. Investors do not need to be as worried about legal responsibilities with Peer-to-Peer due to possible legal troubles only affecting small subsets of users. Overall, Peer-to-Peer is the superior architectural style for this enhancement and will benefit stakeholders more than a Client-Server style.

Maintainability, Testability, Evolvability, and Performance

With the addition of the new submodule, we also introduce new effects on the maintainability, testability, evolvability, and performance of the Apollo system.

Maintainability is defined as the degree to which an application is understood, repaired, or enhanced. With the introduction of the new Cross-Communication submodule, the ease of maintainability of the Apollo system will decrease because of the presence of new dependencies, a new codebase, and new standards to monitor and uphold.

Testability is defined as the effectiveness and efficiency of testing. The Peer-to-Peer implementation of the new submodule requires multiple cars in order to make sure that data is being properly sent and received. Because of the addition of new tests, the amount of testing for the high-level system will increase, but not by a substantial amount since the amount of testing should remain similar to other submodules.

Evolvability refers to the property that programs can easily be updated to fulfill new requirements. The Cross-Communication module can easily adapt and evolve to new changes and features. Our module will introduce an avenue for Apollo developers to push releases to vehicles using the software asynchronously. This means that vehicles will receive software updates in order to be upgraded to the latest software versions.

Performance is the degree to which a software system meets its objectives for timeliness and is related to throughput, response time, and deadlines. The chosen implementation of the module will have an ideal throughput and response time since each vehicle will only be communicating with others located within a certain radius of it rather than with a central server. The response time between other Apollo submodules, however, may decrease since there are new dependencies and more data is being passed between various modules.

The New Feature's Impact on Subsystems

Cross-Communication

The Cross-Communication module would subscribe to: the Perception module for the current object tracking list, HD Map for the current relative map, Monitor for

alerts about module or hardware failures, and Cross-Communication modules of other nearby vehicles for the published data from the Cross-Communication module. The Cross-Communication module publishes: other vehicle's object tracking lists to the Perception module; other vehicle's relative maps to the HD Map module; the status of the Cross-Communication module to the Monitor module so the Monitor subsystem can detect if the Cross-Communication subsystem has crashed; and the vehicle's tracking list, relative map, and vehicle status to the Cross-Communication module of its peers (nearby vehicles). In the event of the Cross-Communication module being alerted from another vehicle that its Guardian module will be forced to stop the vehicle, the Cross-Communication subsystem would publish the other vehicle as an object classified as "caution" to the Perception module.

Perception

The Perception module will be adjusted to now publish the current object tracking list to the Cross-Communication module and will also subscribe to the Cross-Communication module for the object tracking lists received from other vehicles. These other object tracking lists will be merged with the current vehicle's object tracking list created from the various sensors (LiDAR, camera, etc.) used by the car to detect nearby objects. As well, in the event that the Cross-Communication module publishes a message involving another vehicle that has become classified as "caution", the Perception module would add that vehicle to the object tracking list so the car can adjust its route/driving accordingly.

HD Map

The HD Map module will be adjusted to now publish the current relative map to the Cross-Communication module and will also subscribe to the Cross-Communication module for the relative maps received from other vehicles. As well, the HD Map module will now have the relative maps from other vehicles that are further away from the car's current location, allowing it to build a larger relative map, giving the car more "vision" on objects/areas outside of its current range.

Monitor

The Monitor module will be adjusted to now publish an alert in the event of a module/hardware failure, resulting in the immediate stopping of the vehicle, to the Cross-Communication module, similar to the current behaviour between the Monitor module and the Guardian module, and will also subscribe to the Cross-Communication module for its current status.

Guardian

The Guardian module now provides the Monitor module with the type of action it will be taking in the event of a hardware/module failure. This is done since the Monitor subsystem will only have the Cross-Communication module broadcast a message to nearby cars in the event that the Guardian subsystem will be immediately stopping the vehicle as a response to the failure.

Prediction/Planning

While the Prediction and Planning subsystems have not been changed as part of the addition of the Cross-Communication, they are indirectly affected due to the new merged object tracking list and expanded relative map. As mentioned above, the addition of the Cross-Communication module was intended to improve the planning and routing process of the vehicle, yet it does not directly interact with the Planning module. This is because with new objects being added to the tracking list, the Prediction module will be able to classify these obstacles accordingly, which will therefore influence the planned trajectory and route that the Planning module uses. For example, in the event of a car crash on the current planned route, the Cross-Communication module would only make the car aware that this crash exists, not force it to change trajectory. However, due to the crash being labeled as “caution” by the Prediction module, this would influence the Planning subsystem to plan a route that would avoid the crash. Finally, it is worth noting that the object tracking list is shared between Perception modules, rather than between Prediction modules, as the predicted behaviour that would be associated from the object tracking list published from the Prediction module would be based on the location of the car that is publishing the data, not the car receiving it. As such, by assembling the merged object tracking list within the Perception module, it allows each individual car to predict the behaviour of the objects within the list based on their own current localization estimate received from the Localization subsystem.

Impact on Concrete Architecture (Files & Dependencies)

Since the feature implementation requires the addition of a new module and changes to existing modules, there will be a few files and directories that will change as well. As a result of the modified codebase, the concrete architecture itself also changes (see Figure 3).

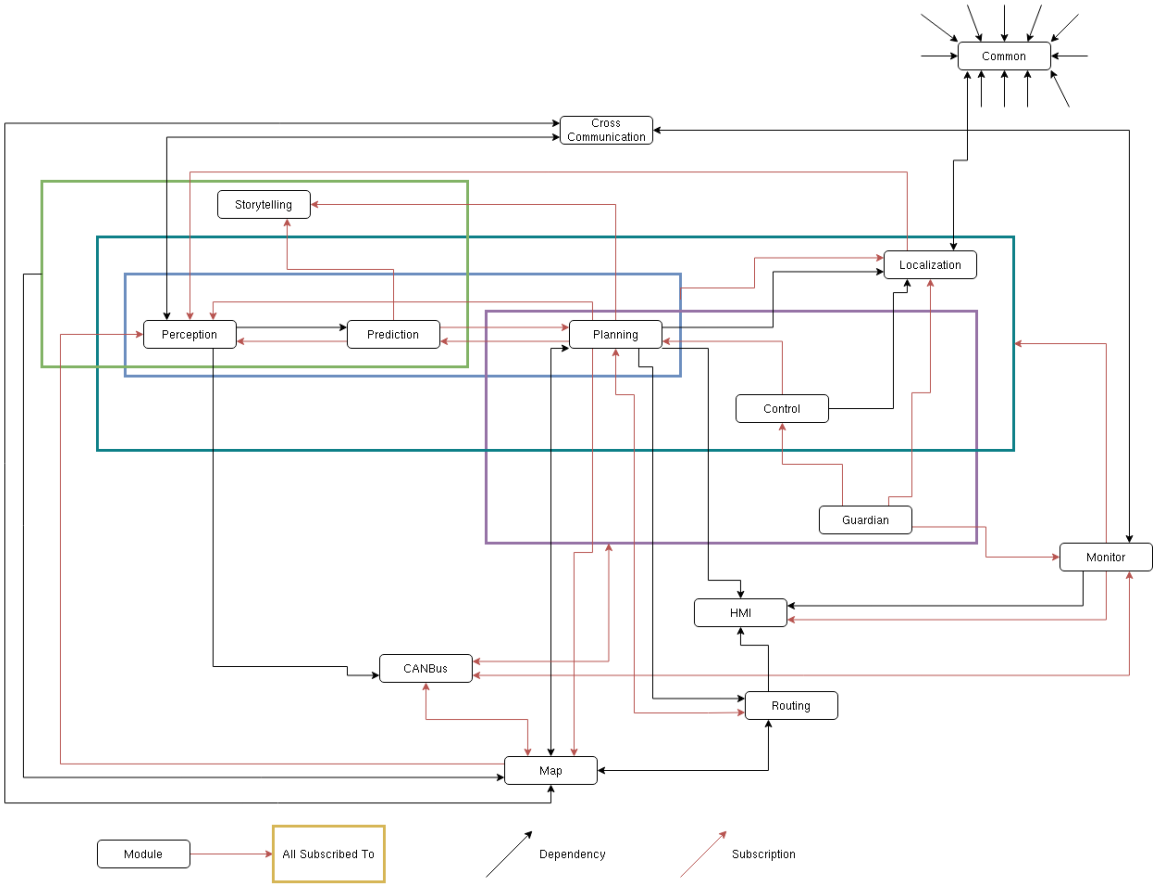

Figure 3. Concrete architecture of Apollo with the new Cross-Communication feature

New Cross-Communication Module

The Cross-Communication module is the main addition for this new feature. It will be responsible for detecting cars within range, and serve as a message broker that uses a special communication protocol in order to send and receive messages. Since this is a new major module, it will be given its own “cross-communication” folder under the “modules” folder in the Apollo repository. Like all of the other main modules in the concrete architecture, it will be given a “Base” component which contains any common methods used by the rest of the cross-communication module. The Base component will also serve as the main interface for other Apollo components. It will be subscribed to any information coming from the Perception, Map, and Monitor subsystems. Apart from the Base component, there will also be a dedicated directory for Peer-to-Peer protocol specific methods. This includes methods for sending and receiving messages via the chosen protocol. Cars also need to install hardware in order to send and receive messages from other cars over-the-air. A dedicated “hardware” component in the Cross-Communication module will also be necessary to configure this hardware. Finally, there should be a “testing” folder, responsible for containing any unit or integration tests for the Cross-Communication module. This component should be used by developers to ensure that this module functions as intended.

Perception

The Perception module should be updated to handle the serialization of data it

needs to send to other cars, as well as deserialize the information that it receives back from other cars. Since the serialization and deserialization of data is specific to the operation of the Peer-to-Peer system, a dedicated set of code will be added into a “cross-communication” folder, under the dedicated “perception” folder. The Perception module will also include a new sensor type, called “P2P”, along with the normal camera and LiDAR sensors. These are located in the “drivers” folder. Surrounding cars can be interpreted as sensors for each other. This code will be responsible for tagging objects detected by other cars in a way so that other Apollo modules, like Planning and Prediction, can understand that the information came from other cars. Finally, the Fusion component will also be changed to aggregate the data from the other cars as well. This involves creating a new file under the “fusion” folder. After the data from other cars are deserialized and tagged appropriately, the car should aggregate, or fuse this information with the data from its connected sensors. The combined data is what is finally published to the rest of the Apollo system.

Monitor

The Monitor module will need to be able to track the state of other cars as well. In addition to tracking the state of other cars, the Monitor system should also send the status of the current car’s modules to other cars. If the current car experiences a failure, other cars should be aware of it. The main change here will be in the Base component that includes all of the common methods in the “monitor” folder. Particularly, a new set of files should be added under the “software” directory, that allows the Monitor module to examine the state of other cars. If a nearby car were to fail, this file should be responsible for sending a command to the Guardian module to react accordingly (i.e. steering out of the way). Apart from modifying the Base component, there should be a new “cross-communication” folder. It will have the same functionality as outlined previously for the Perception module. That is, it will be responsible for serializing the state of Apollo modules, and deserializing the data coming from other cars.

Map

As new object data comes in from surrounding cars, the current car should be able to expand its HD Map and relative map accordingly. Under the “map” folder, there are directories for the “relative” and “hd” maps respectively. After receiving data, these submodules should be able to populate themselves with the newly detected objects, and also send this information to other cars. As always, a specific “cross-communication” folder will be needed to serialize and deserialize this data when sending or receiving messages from other cars.

Prediction

As mentioned previously, this module does not directly communicate with the “cross-communication” submodule and will be affected indirectly. Updated predictions for object trajectories will mainly be handled through updated Perception and Map data. However, the prediction logic will need to be updated in order to account for the potential unreliability of the data obtained from the Peer-to-Peer system. For example, if a nearby car is just barely within range, it may experience increased latency when sending or receiving data. The code should handle this by, for example, ignoring P2P-tagged data that has an old timestamp. In particular, the Prediction module should handle these things by updating its Evaluator and Predictor components. These components can be found under their respective folders in the “prediction” module

folder. New AI models may be introduced to evaluate, and predict the future motion of objects obtained through the Peer-to-Peer system. After the Prediction module accounts for the new data, it will send its data to the Planning module, and continue with the usual Apollo data flow.

Potential Risks and Limitations

With the implementation of a new module in the Apollo system, there are various risks associated with it. The new Cross-Communication module introduces three potential risks, as a result of both the module itself, and its interactions with other modules already in the Apollo system: security, scalability, and performance.

Security

The Cross-Communication module functions in a Peer-to-Peer system, wherein vehicles communicate directly with one another. Security vulnerabilities in the Apollo system arise with the implementation of the proposed module, with the possibility for users with malicious intent to modify the data transfer mechanisms, possible since the Apollo system is open-source. Ultimately, development of the Cross-Communication module must make note of the specific modules that should be affected and that can receive data. For example, the Control module must not be provided access through the Peer-to-Peer connection. Beyond this restricted access to the system, data transferred to a peer vehicle should be validated to be real, as the transferring of incorrect data could also be detrimental to the functioning of the vehicle and its planning. In a real world application, disabling the Cross-Communication module would eliminate these security vulnerabilities, though would also reduce overall network coverage by removing a peer from the interconnected roadway system.

Scalability

Using a Peer-to-Peer system, the Cross-Communication module may directly interact with and share data with other nearby vehicles also using the Apollo system. On a populated roadway with many Apollo-based vehicles, Apollo systems would be able to gather a lot of information to better plan situations and ensure the safety and security of passengers. Scalability concerns arise with the amount of data to be transferred; under large amounts of traffic where a chain of Peer-to-Peer communication would arise, the amount of information and data sharing may be problematic for vehicles to gather, and for the Cross-Communication module to appropriately process the data to effectively use in the Apollo system. Instead, the module would need to assess data within a certain vicinity, limiting the effectiveness of the Cross-Communication module in gathering distant data that could be useful in the planning of the system (e.g., traffic accidents down a road, which may influence routing of the vehicle).

Performance

Besides the potential scalability issues arising from the Cross-Communication module, the Apollo system may also suffer severe performance issues. With the new data provided from other peer vehicles, while it may allow for enhanced operation of the vehicle, it would be computationally challenging. In the situation of retrieving data shared in traffic, processing of the data could be difficult for the Apollo system and

hardware to perform in an efficient manner because of the sheer amount of data transferred, therefore potentially being bottlenecked by the hardware of the Apollo system. This situation of a bottleneck could be detrimental to computing important information, especially in relation to the control of the vehicle to protect the passengers in it.

Testing

To test the system, it is imperative to test the impact of interactions between the proposed enhancement (the Cross-Communication module) with the pre-existing functionality of the Apollo system. A combination of unit testing during the development of the new module, as well as integration testing during implementation of the module into the broader Apollo system will ensure that the Cross-Communication module functions as intended, mitigating bugs and errors in the module.

Unit testing involves verifying the functionality of the new features, prior to integration into the system. For the Cross-Communication module, its data transfer mechanisms and the selected data that is shared between two peers should be examined during testing to ensure that the data transfer is secure and effective under various circumstances. By first testing the module itself prior to integration into the Apollo system, intra-module errors can be more easily identified, while simultaneously ensuring that the module satisfies its functional and non-functional requirements.

Integration testing involves examining and validating the functionality of the new feature in the broader system. In the context of the proposed feature, integration testing would entail examining the interactions between the Cross-Communication module and the pre-existing modules in the Apollo system. Due to the nature of the Publish-Subscribe structure of the Apollo system, the integration of the Cross-Communication module should have minimal effect on the other modules, and should not drastically change pre-existing functionality negatively. Importantly, the Cross-Communication module should only influence the rest of the system through the data it provides from other peer vehicles. To be specific with integration testing, testing can demonstrate the use of the new Cross-Communication module, and consequently its new data, to aid in specific driving scenarios.

Uses Cases and Sequence Diagrams

To demonstrate the Cross-Communication feature, we can explore two use cases. These two use cases cover generally what happens when the user vehicle is receiving a message from a peer vehicle, and what happens when the user vehicle is sending a message to peer vehicles, respectively.

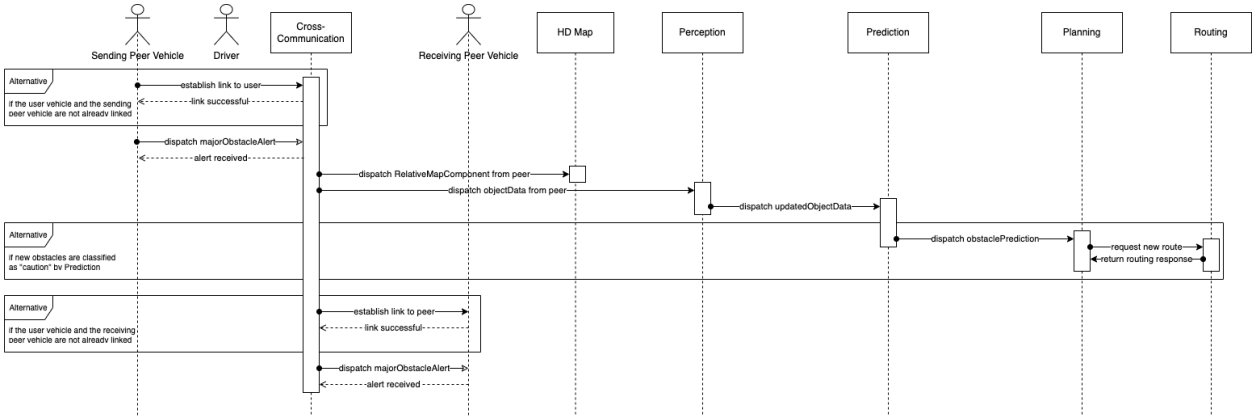

Figure 4. Sequence diagram for a use case involving a major obstacle perceived by a peer vehicle

The first use case (Figure 4) depicts the communication that occurs when a peer vehicle (referred to as "Sending Peer Vehicle" in Figure 2) of the user perceives a major obstacle. The peer vehicle's own Cross-Communication module, which is an Apollo vehicle in near proximity to the user vehicle, attempts to establish a link to the user vehicle's Cross-Communication module if the link does not already exist. Once the link is established, the peer vehicle alerts the user vehicle via their respective Cross-Communication modules that the peer vehicle has detected or been alerted by another vehicle of a major obstacle. The user vehicle's Cross-Communication module responds by confirming the alert has been received, then begins action on this alert internally. This action first starts with updating the relative map within the HD Map module, accomplished through Cross-Communication, which publishes the peer vehicle's own relative map, allowing the user vehicle to maintain an internal map even beyond its own perceiving scope. Cross-Communication then publishes the data about the major obstacle to the Perception module, which adds the obstacle to its object tracking list, which itself is published to the Prediction module. The Prediction module then estimates the behaviour of the major obstacle relative to the user vehicle, including assigning a priority of either "ignore," where the obstacle can be safely ignored, "caution," where an obstacle has a high possibility of interfering with the vehicle's trajectory, or "normal," the default priority value in which the obstacle falls under neither of the prior two priorities. The prediction for the major obstacle is published to the Planning module, along with the other obstacle predictions. In the case the major obstacle is classified with the "caution" priority, the Planning module publishes a new routing request to the Routing module, which then generates a new route for the user vehicle to avoid the major obstacle. Once the user vehicle has finished taking internal action, it itself sends an alert regarding the major obstacle via Cross-Communication to other peer vehicles (referred to as "Receiving Peer Vehicle" in Figure 2). If not already linked with Apollo vehicles in near proximity, it will first establish the link, then dispatch the alert, triggering the same internal action sequence seen in the user vehicle in the receiving peer vehicles.

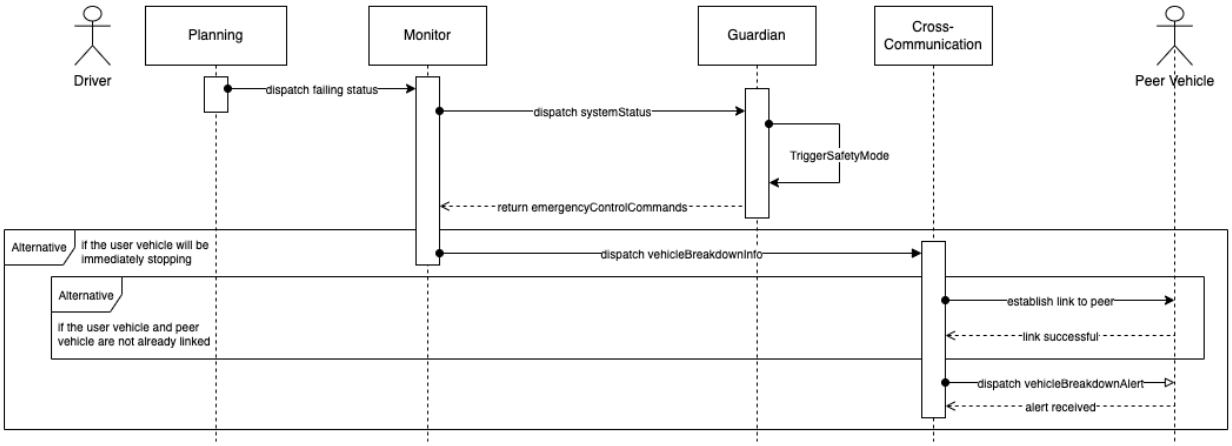

Figure 5. Sequence diagram for a use case involving failure of the user vehicle's Planning module

In the second use case (Figure 5), we explore what the new feature does when the user vehicle itself could become an obstacle. This occurs when the user vehicle is experiencing module or hardware failure, in which case the Guardian module must take over to safely stop the car. It may be difficult for nearby Apollo vehicles to anticipate the Guardian module's emergency commands, so the user vehicle's Cross-Communication module takes action to alert them. Before it can do so, the internal course of actions where the emergency commands are actually enacted must take place first. In this use case specifically, we look at the scenario where the Planning module crashes. It publishes its failing status to Monitor, which then publishes the system status to Guardian. From this data, the Guardian module detects that the system is experiencing module failure, so it triggers safety mode, which allows it to override the commands published from the Control module. Depending on the scenario, the vehicle will either be brought to a slow or immediate stop. The emergency commands are shared with Monitor, and if the vehicle is being brought to an immediate stop, Monitor will publish this information to the Cross-Communication module. If a nearby Apollo vehicle is not currently linked to the user vehicle via Cross-Communication, the module tries to establish a link. Once established, Cross-Communication can alert all its peer vehicles that the user vehicle is experiencing failure and will be coming to an immediate stop, enabling the peer vehicles to make alterations to their own trajectories in the case the user vehicle itself becomes an obstacle for them.

Conclusion

To conclude, the improvement that we propose is Peer-to-Peer communication between vehicles operating using the Apollo system. Initially examining two different approaches to implement the feature—a Client-Server architecture versus a Peer-to-Peer module—we concluded that the latter would be ideal for the feature's implementation into the Apollo system. Our team came to this conclusion by performing an SEI SAAM analysis on both implementations of the Cross-Communication module: examining the key stakeholders, respective non-functional requirements, and the impact of the addition on the identified points. We found that the Peer-to-Peer module would allow Apollo vehicles to form Peer-to-Peer connections, exchanging collected data and sharing with its peers, and thereby affecting subsequent computations for the planning

of the vehicle and different scenarios. With the new module in mind, we updated and further examined its impact on the concrete architecture which we previously established using reflexion analysis. With the addition of the Cross-Communication module, we saw major changes in the interactions with the Perception, Monitor, Map, Prediction, and Guardian modules. Ultimately, while the proposed feature is meant to improve the Apollo system, we also identified risks and limitations related to the security of the Peer-to-Peer clients, the scalability of the data transfer mechanisms, and the performance of the current Apollo system using collected data. To mitigate some of these risks, it was deemed essential to adopt unit and integration testing techniques during the development of the module and leading into deployment. Lastly, with a deeper understanding of the proposed functionality, we examined specific use cases of the feature, summarizing the data transfer and retrieval mechanisms of the new Cross-Communication module.

Lessons Learned

As we investigated the different implementations of our new feature, we learned of the importance of communicating the differences between various implementations. We learned that in order to reach the most appropriate solution, we needed to consider the advantages and disadvantages of each. By performing an analysis on the different approaches, we found that small changes in the architecture of our solution could have big implications. Rather than immediately deciding on one solution, an investigation into alternatives helped us come to the most sensible solution for the problem that we want to solve.

In addition to performing an in-depth comparison between implementations, we also discovered the importance of continued discussion and communication between our team. When discussing the implementation of our new feature, we found that different members had varying ideas, so it was important to come to an agreement about how we decide to execute it. This helped us gain a detailed understanding of the overall solution, and kept our team fully aligned with the solution. Lastly, the continued communication allowed us to come up with solid goals and deadlines for us to achieve. Creating goals and deadlines kept us on track, and ensured that we accomplished everything that we needed to.

Data Dictionary

Autonomous vehicle: A vehicle that can drive and perceive its environment without a human controlling it.

Client-Server: An architectural style involving communication between a server and clients through a centralized network.

Conceptual architecture: A very high-level organizational view of a system, highlighting modules and the relationships between them.

Concrete architecture: The structural make-up of a piece of software, containing modules and their interconnections

Major obstacle: An unexpected obstacle on the road that could cause delays while driving such as a car crash, a dangerous driver, a traffic jam or car pileup, construction, an animal, or other large debris

Module/Subsystem: A bundle of code composed of many related functions contributing to a common feature.

Open-source: A code-base that is publicly accessible to be used or contributed to.

Peer-to-Peer: An architecture style involving the communication between stand-alone components (peers) that share data and workload without centralized control.

Peer vehicle: An Apollo-operated vehicle within a user vehicle's communication range, which is dependent on hardware and software capabilities, requiring testing to figure out the optimal range

Publish-Subscribe: An architecture style involving loosely-coupled collections of modules. Each module is responsible for an operation, which may enable other modules in the process.

Naming Conventions

LiDAR: Light Detection and Ranging

NFR: Non-Functional Requirement

SEI SAAM: The Software Engineering Institute's Software Architecture Analysis Method

P2P: Peer-to-Peer

References

GitHub, Inc. (2017, July 2). *Apollo's GitHub Page*. GitHub. Retrieved March 4, 2022, from https://github.com/ApolloAuto/apollo

Montgomery, M. (2020, February 7). *How many days of your life are you losing stuck in traffic?*. Rcinet.ca. Retrieved March 4, 2022, from https://www.rcinet.ca/en/2020/02/07/how-many-days-of-your-life-are-you-losing-stuck-in-traffic/#:~:text=A%202019%20Statistics%20Canada%20study,of%20work%20was%2057%20kilometres