Distributed Transport Network

Brandon Thai Nguyen, Daniel Smith, Tyler Goffinet
05 December 2016
CPE 400.1001 – Project Report

Introduction

With innovations in technology rapidly evolving every day, autonomous vehicles will eventually become the prominent drivers on the road; it is becoming important that vehicles are able to communicate statistical data with other vehicles. This statistical data can include anything from speed to location and destination for applications such as analyzing traffic congestions, calculating safest routes to a destination, and determining the most optimized placement of cars on each lane of any given street all potentially maximizing flow of traffic and safety. Communication also should not be limited to cellular and satellite communication since circumstances exists where those communication methods are limited so it is wise that vehicles also have the ability to transmit data within a short range through a network of multiple vehicles in range via a peer-to-peer mobile protocol. Our team has created peer-to-peer protocols to effectively transfer packets of information allowing for the routing packets within a network of vehicles. In order to test the effectiveness of this protocol, we created a program to simulate such a network in which our protocol yielded promising results.

Simulation Structure

To start, our team needed to build a simulator program before creating any peer-to-peer protocols to be able to test them rigorously.

Simulator World

The world class of the simulator is kept relatively simple to avoid unnecessary debugging when designing new protocols. The world is constructed as a grid style city with equal distances between each adjacent intersection. The world is represented as a two-dimensional array of vehicle class pointers with the length and width defined by the user. Every array element is defined as an intersection and the roadways between intersections are simulated via a timer representing the travel time between each intersection. Having the world represented as a two dimensional array has the advantage of having array elements that are set to NULL signifying that there is no vehicle present in that intersection.

The simulator is also kept simple by running the world off a tick-based timing system. Each vehicle operation such as moving and sending packets is set to perform their tasks in an arbitrary number of ticks. The tick-based timing system was implemented to keep a level of realism; vehicles are not going to move as fast as a packet in the real world.

The world handles the vehicle movement and running the vehicle's functions within the world class. This allows the world to be in control of the vehicles itself and act as the bridge between multiple vehicle objects.

Vehicle Behavior

Moving forward, the simulator needed a system to handle vehicle behavior. To begin, vehicle rules are set in place for the vehicle behavior. First, there cannot be more than one vehicle in an intersection at a given time, second, vehicles can pass through each other even though each intersection can only contain one vehicle, and third, the vehicles are not allowed to turn around immediately. These rules were set in place strictly for simplicity as the main focus of the project is on the routing protocols.

Additionally, the vehicle behavior is controlled by an AI responsible for the vehicle's destination, routing to the destination, and potential collisions with other vehicles.

First, the AI calculates the destination for the vehicles in very simple manner. It is done with a random number generator that outputs a random coordinate within the range of the world.

Second, the AI handles the vehicle routing getting the vehicle to its calculated destination. The vehicle will tell the world class to move itself row-wise until they are in the same column as the destination. Afterwards, the vehicle will have the world class move itself column-wise until the destination is reached.

Third, the AI is responsible for handling collisions. If the path is obstructed, the vehicle will attempt to go in an optimal direction perpendicular to its current direction. If the path is further obstructed, the vehicle will attempt to wiggle its way out by trying all different directions unless there is no free direction which then the vehicle will wait.

Overall, the vehicles behavior defined by a set of rules and controlled by an AI that determines where the vehicle will go.

All in all, the simulator plays a big role in the testing of protocols we create since it simulates a simplistic grid city world that is populated by moving vehicles with a behavior system.

Routing Algorithms

We created two algorithms for this project. The first was designed as a basic flooding algorithm to act as a baseline protocol. The second protocol, destination search protocol, we created as in intelligent routing protocol.

Flood Algorithms

The flood algorithm has each vehicle broadcast any packets it may contain to all vehicles in its transmission radius. Once a vehicle successfully throws its packet to at least one vehicle, determined by a response from the target vehicle, it will not throw that packet again. After five cycles the packet will be discarded. This algorithm works well in high density areas but is costly as each vehicle must occupy time dealing with a packet even if that vehicle cannot move the packet any closer to the destination.

Destination Search Algorithm

The destination search algorithm uses the recipient's vehicle destination as a method for routing the packet. For this algorithm, each vehicle must maintain a list of where other vehicles are traveling to and from. It would be far too costly to have each vehicle send out its current location at each cycle so instead, we have each vehicle output its next destination and the location the vehicle is starting from. This information is shared by transmitting an update packet with the vehicle's new destination and starting location. The update packet is sent whenever a vehicle reaches its destination and recalculates a new destination. This packet transmission is sent to all vehicles in the system via the flood algorithm. Upon receiving an update packet, the receiving vehicle list of vehicle destination and starting locations will be updated. This list will allow vehicles to find a smarter way to locate the packet target.

For a packet being transferred from Vehicle A to Vehicle B, the following steps will follow: Vehicle A will first check its list of surrounding vehicles to see if Vehicle B is within range and if that is the case, Vehicle A will throw the packet to Vehicle B. If Vehicle B is not in range, then Vehicle A will look at its array of vehicle locations to see if it has any information on Vehicle B. If Vehicle B is in this list, then Vehicle A will take Vehicle B's destination and compare it to Vehicle A's current location to determine which direction would be ideal to attempt to throw the packet for it to intersect with Vehicle B. Once Vehicle A has an ideal

direction it cycles through all vehicles in its immediate vicinity and assigns each one a score signifying how ideal it would be to throw its packet to that vehicle. The score is calculated off the other vehicle's current direction and its position in relation to the ideal direction to throw calculated earlier. Vehicle A also gives itself a score to determine if the best choice is to not throw the packet at all. After all vehicles are considered, the packet is thrown to the vehicle with highest score. After the throw, the process repeats with the new vehicle containing the packet. The results showed this algorithm in its current form underperforms but with proper optimization may perform better.

Results

Once the simulator and algorithm were created we put them through a series of rigorous tests to gauge the effectiveness of the algorithms.

Testing Method

For testing purposes, we used a command file to generate 100 packets in a 10 X 10 world with varying levels of vehicle density. We tested with three levels of density: low (10 vehicles), medium (20 vehicles), high (49 vehicles, the maximum allowed by our simulator). We gave each packet 100 ticks to reach its destination, any packet not reaching their target in that time are considered to have failed. The two algorithms preformed quite differently as can be seen below.

Flood Algorithm

The flood algorithm performed better as the world became more densely populated. The reliability of the algorithm behaved as expected: decreasing when the population got sparse and increasing when the population got dense as in figures 1, 2, and 3. In addition, the overhead for our flooding algorithm increased as the number of packets transmitting also increased. This is due to the amount of unnecessary packet transfer flowing through the network. While the overhead increased it still performed faster than our destination search.

Destination Search Algorithm

The Destination Search algorithm had only a ~60% success rate, however, the average number of ticks to achieve packet transfer was less than 5.

We hypothesize the low success rate of the destination search to the weights used to determine the best vehicle choice. The vehicle holding the packet becomes trapped by other vehicles with low calculated scores, the packet then becomes trapped with the vehicle.

While running our tests we also noticed that our code contained a serious memory which while effecting the physical run time of the simulation did not impact the performance of the simulation itself.

The algorithm performed most optimally in networks with low population density because it was able to take full advantage of vehicles movement. In high density networks as the vehicle traffic became more congested and vehicles moved less freely the chance of the packet remaining with the current vehicle increased, thereby decreasing the efficiency of the algorithm. This is one of the major factors contributing to the algorithms lack luster performance compared to the flood algorithm.

Comparing the Algorithms

When comparing the two algorithms it can be seen that the performance of the Destination Search algorithm is not what we hoped it would be. Table 1 shows that the flood algorithm is better in terms of percentage of successful packets, number of ticks it takes to transmit a packet and has a smaller standard deviation of transfer times. The only saving grace of the Destination search algorithm can be best seen in Figure 2 where we can see that the time to complete had a downward trend as the destinations of packets permeated the network. This downward trend is a good indication that this algorithm can be modified to become a good candidate for transmitting data in a mobile ad-hoc network.

Conclusion

Our team was successfully able to design and implement an algorithm tested in our own simulator environment solving the problem of close range peer-to-peer communications between multiple autonomous vehicles. Hopefully future autonomous car makers can have a broad set of communication tools to allow vehicles to transmit vital information from one vehicle to another to opening up doors for various types of applications like safety, analytics, and more. Even though our team has successfully created a custom protocol that handles peer-to-peer communication between a dynamic network of moving vehicles, we would have liked to try out other algorithms already made such as VANET, CASNET, DSR, BATMAN, AODV, and others of the similar sort. We also would have even liked to test out our protocols using real hardware provided by the CSE department but could not due to time constraints. Overall, our team has learned a lot about different routing algorithms and protocols as well as having exposure to out of the box thinking creating a protocol to handle a situation that may appear as autonomous vehicles become more prominent.

	Flood	Destination
Average Ticks to complete	4.670881	4.890547875
Completion rate	84.33333	60.66666667
Standard Deviation	2.933752	4.608963352

Table 1: Shows the areas in which Flood algorithm outperformed the Destination Search Algorithm

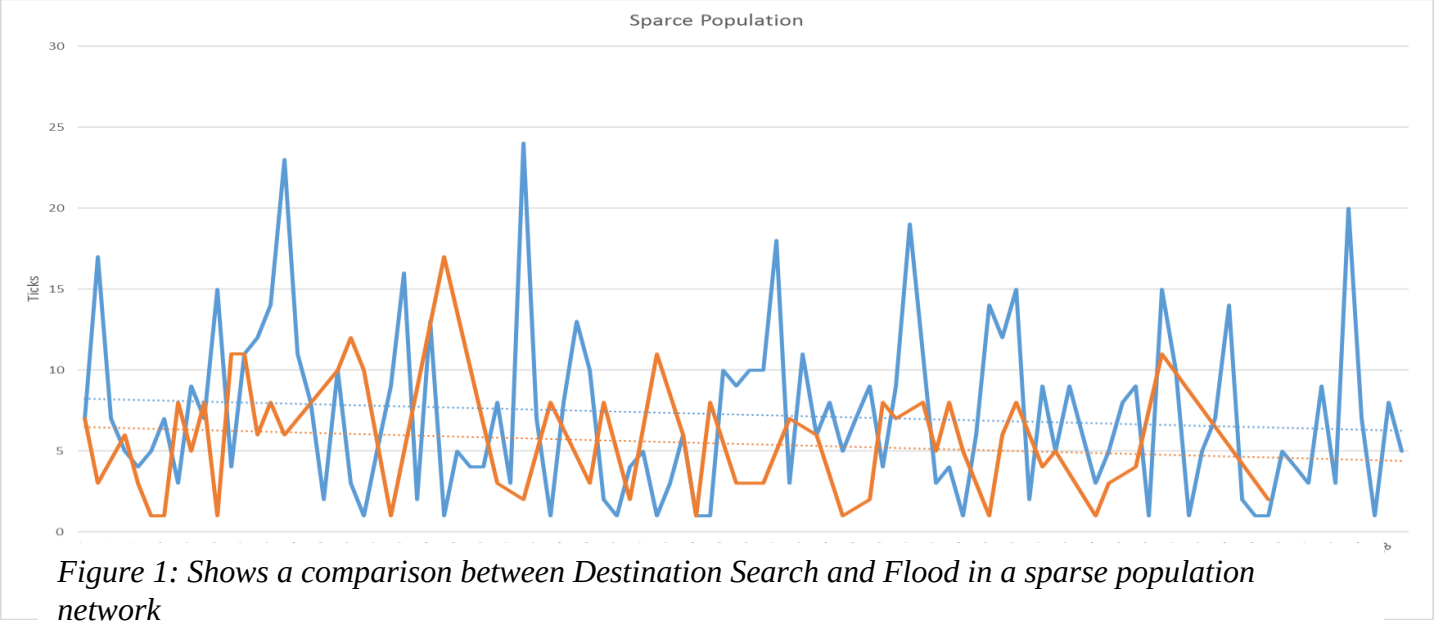

Figure 1: Shows a comparison between Destination Search and Flood in a sparse population network

Figure 2: Shows a comparison between Destination Search and Flood in a Moderate population network

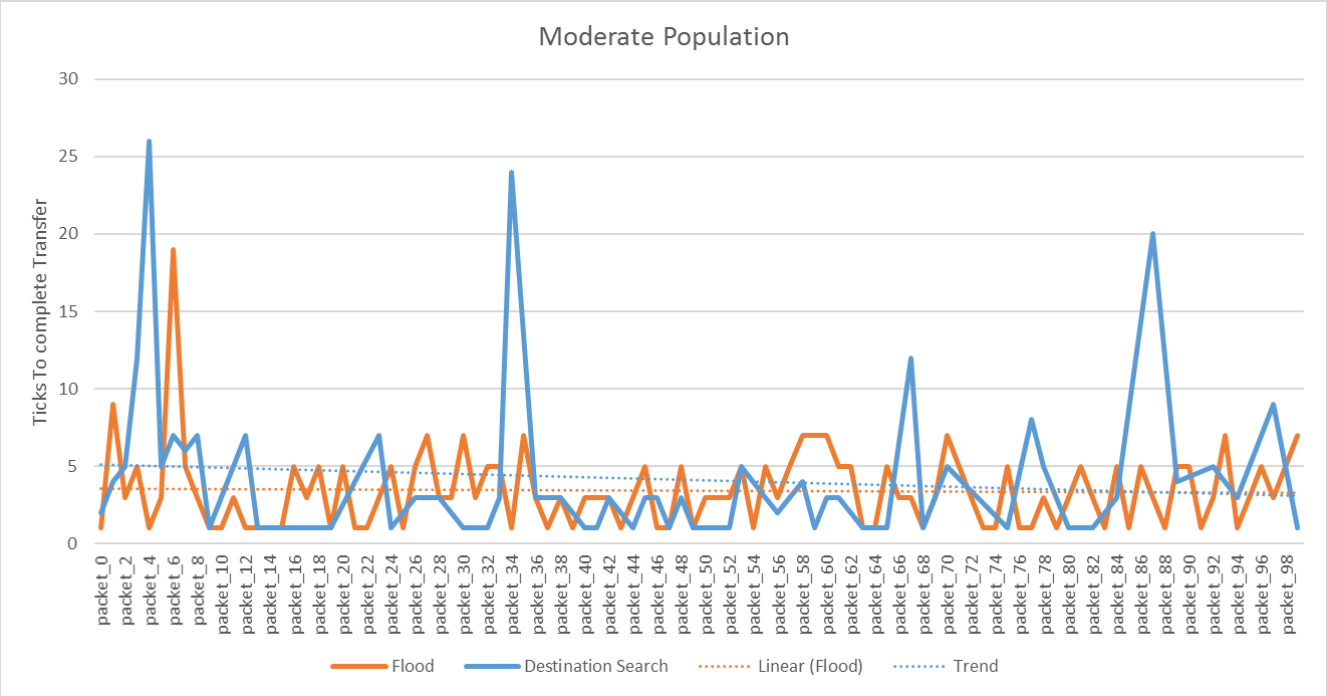

Figure 3: Shows a comparison between Destination Search and Flood in a sparse population network