Implementing Road Train with Reliable Broadcast Algorithm in Vehicular Ad-Hoc Networks (VANETs)

COMP 5360

March 18, 2015

Andrew K. Marshall

Evan Hall

Prashanth Tanjore Saikumar

## INTRODUCTION

The Advancement of Mobile Ad-hoc Networks has opened avenues for vehicular communication, with vehicles acting as wireless nodes to form a mobile network. This mobile network is termed as Vehicular Ad-hoc Network (VANET). VANETs were developed with the intent of making transportation safe and convenient. Cars companies of the future may be mandated to install VANET systems in all their vehicles due to the potential safety and economic advantages such a system brings. As promising and exciting as this technology may appear, it is not lacking in complications. This project illustrates some of the challenges and issues of implementing a Road Train that uses the VANET technology.

## PROBLEM STATEMENT

Implementing a VANET to make an automated Road Train possible has many intrinsic challenges. Data packets must be constantly and reliably transmitted to and from every single car, the bandwidth is limited, speed is essential, and the system must be reliable enough for people to trust it with their lives. In this project we focus on the reliability aspect and see how reliability can affect overall performance.

## OBJECTIVE

We aim to design and implement a reliable broadcast algorithm over a vehicular ad-hoc network (VANET) that will support the implementation of a Road Train cooperative adaptive cruise control system. In pursuit of this objective, we use PCs to act as mobile nodes (a vehicle in this case) that will be used to model a Wi-Fi environment. We intend to replicate a wireless vehicular ad-hoc network and allow it to transfer data packets using a reliable broadcast protocol. This exchange of packets will enable the nodes (cars) to safely navigate the highway and to successfully form a platoon.

## ARCHITECTURE

Our program is comprised of three threads:

* Server thread
* Main thread
* Timer thread

The server thread opens a UDP server and listens for incoming packets. When a new packet arrives, it is pushed onto a queue that the main thread can then pull from. We use a mutex to ensure a packet cannot be pushed to the buffer in the server thread while the main thread is pulling from it.

After its initial set up, the main thread goes into an infinite loop that constantly checks the packet buffer. Whenever a packet exists in the buffer, it is retrieved from the front of the queue and processed. Packet processing involves running it through the reliable broadcast algorithm and forwarding it as necessary. These packets are also used to update other node’s positions and determining if said node needs to slow down, change lanes, join a platoon, etc.

The Timer thread waits for a set amount of time and then sets a Boolean variable to true. Our main thread will transmit a status packet to all its neighbors only if this Boolean is set to true. After a status packet has been sent, it sets the Boolean to false. Like the server thread, a mutex is used to prevent simultaneous access to said Boolean. This allows us to constantly process packets in the main thread and only send status packets at regular intervals.

## DESIGN

The following subsections discuss design details and program implementation.

### Simulation set-up

Upon startup, the program checks if a Configuration file exists. If the file does not exist, the program will create a Configuration file and in-turn will know that it is the first node to join the simulation and therefore is the truck. If the file does exist, the node is designated as a car and reads in existing node information from the Configuration file. Once the node reads the Configuration file, it sends out an initialization UDP packet that indicates to the other programs that a new node is joining the simulation. Then the joining node will wait and listen for status packets. Once it has received a status packet from every node, it can safely find a place to enter the highway.

### Updating the positions of every node

When the timer thread indicates that it is time to send a status packet, each node prepares a status packet containing its current location. The current node will then iterate through a list of all the other nodes in the simulation. If two nodes are in range of one another (100 meters), then we calculate the probability the packet will be sent/received successfully. If the random number generator indicates the packet will be sent, then the status packet is sent and the receiving node will process it like normal. However, if the two nodes are not in range or if the random number generator indicates the packet was lost, we set a flag in the status packet, but send it anyway. This packet is labeled as an “Admin Packet.” When an Admin Packet arrives, the receiving node will use that packet to update its data structure that keeps track of every other node’s position; the packet, however, is not processed further. Admin Packets never go through the Reliable Broadcast Algorithm. We decided to use this approach to update the node’s locations rather than using Configuration file reading/writing. This is discussed further in the “Implementation Issues” section.

### Writing the output data

Apart from the standard output text you see on the screen, the truck node will write to an Output.txt file the current locations and connections of every node. This allows the user to view the current locations of every car at any point during the simulation by typing “cat Output.txt” in the terminal.

## ALGORITHMS

### Reliable Broadcast Algorithm

The main purpose of this project was to implement the Reliable Broadcast Algorithm. Since the algorithm details are detailed in the project specifications, we will not discuss it in depth in this report. However, the general idea was to flood the network with duplicate packets in order to ensure maximum reliability. We will see in the “Results” section how this affects performance.

### Road Train Algorithm

The Road Train Algorithm was designed with two main goals in mind. One, to ensure cars can safely navigate the highway without crashing into each other (Road Rules); two, to enable and maintain platoon formation behind a lead truck (Road Train Rules).

The Road Rules are as follows. Cars always want to be in the right lane whilst maintaining their current speed. If they get a packet indicating a car is less than 20 meters ahead, they will check and see if the left lane is clear. If the lane is not clear, they simply slow down and keep checking the left lane. Provided the left lane is clear, they will change lanes and attempt to pass the slower car located in the right lane. Should they get too close to another car in the left lane while attempting to pass, they simply slow down until it is safe to speed back up or get into the right lane again.

In order for a car to join the Road Train, it must first place its request with the lead truck. Upon receiving this request, the truck checks to see if it is in the middle of an ongoing link establishment with another car. If this is the case, the truck will not progress further with this request and sends a packet to the car telling it to wait. Otherwise, the truck will respond with information about the car’s platoon order number. This information tells the car how far behind the lead truck it should always be. Once the car is in position, it sends an “All Clear” packet to the truck indicating that new cars may join the train.

To see more details on the Road Train Algorithm, please see **Road Train Algorithm**.

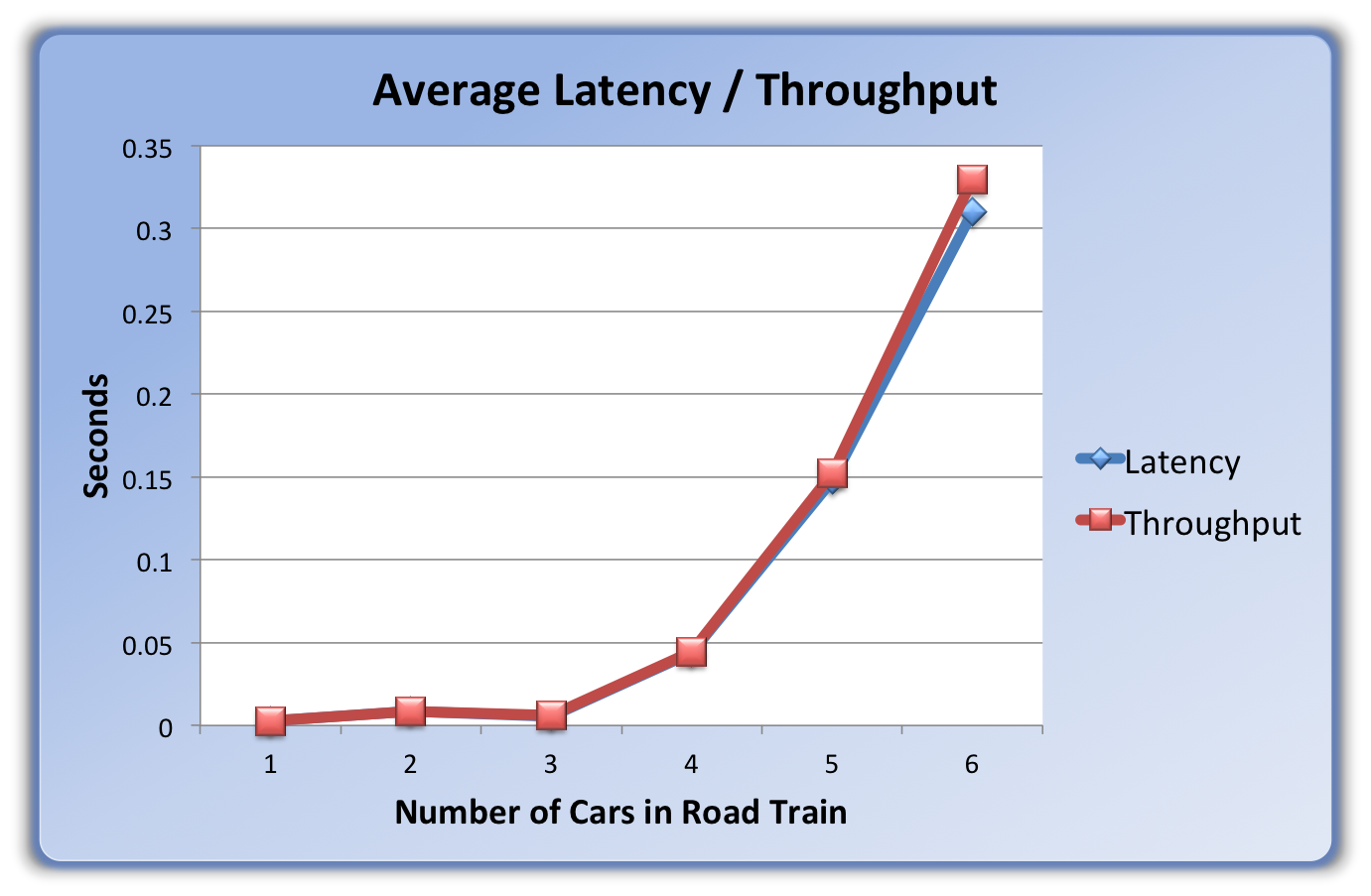
## Implementation Issues

Our biggest issue was dealing with continuous file reading and writing. The file writing issues were manageable when using a computer with a single core, but once we ran it on a multi-core system, the problems caused were just too severe. To resolve this issue, we moved away from our reliance on file reading and writing to a more reliable Admin Packet system discussed in **Updating the positions of every node**. This Admin Packet system reduces Configuration file access to a single write by a Car node, occurring when that node is first joining the simulation. Other than that, the Truck node does all the output file writing.

Secondly, we have issues with timing. We have a function that returns the current time. However, we sometimes see rare cases when it computes a negative difference of time from some previous timestamp. This caused us to have to throw out some of out response time and turnaround time data, as seen in the included spreadsheet with all our experiment data.

## Results

In order to better understand the performance of the Reliable Broadcast Algorithm, we ran a multitude of simulations simulating different scenarios. Our scenarios range from one truck and one car to one truck and six cars. We were unable to collect meaningful data after the sixth Car node, as the program could not handle the network traffic while still be responsive enough for the simulation to run smoothly.

All of our experiment data can be found in Appendix B, and the Excel File submitted with this report.

### Latency and Throughput

One requirement of any VANET system is that it needs to be fast. If a car unexpectedly applied the brakes ahead of your vehicle, you would want as much time to react as possible. So naturally, we wanted to see how latency and throughput time compared for different scenarios. What we found was that as long as the network traffic was kept to a minimum, the response and turnaround time was very small. However, once more cars started joining the network traffic increased and caused the response time to increase dramatically. One interesting note was that once the packet was received, it was processed almost immediately. Therefore, the bottleneck was not with the processing time, but with the packet buffer filling up too quickly.

### Packet Loss

As with any wireless system, packet loss is always a constant concern. In our experiments, we found that packet loss increased with the number of cars in the road train. However, this graph fails to explain the whole situation. From the graph, it appears that the packet loss would increase linearly with the number of cars in the road train. In practice, however, this graph would likely plateau after six or seven cars. This is because each node can only transmit 100 meters. Once there exists seven or more cars in the platoon, they remain in range of the same amount of cars continuously. Therefore, packet loss would not increase but stay constant. The reason why Road Trains with only one or two cars have such a low packet loss is that once the road train “settles,” those cars are never really that far away from one another. They stay at a relatively short distance apart and therefore have greater chance of successfully transmitting to their neighbors.

Another interesting observation regarding packet loss is that the lead truck (and the rear car in a long platoon) tend to have the greatest packet loss. This is caused by the fact that the nodes on the end can only transmit to half of their effective area. The truck is only in range of the cars behind it and it only takes a few cars before the truck is transmitting at its maximum range. This is why a packet-forwarding algorithm is so important.

### Redundant Rebroadcasted Packets

A major characteristic of the Reliable Broadcast Algorithm is that many of the same packets tend to be forwarded more than once. If a node receives a packet it has already forwarded it will forward that packet a second time with a 50% probability, then a 25% probability and so on. That being said, the majority of packets traversing the network are redundant. To get an idea of how much of the network traffic is redundant, we kept track of how many packet were rebroadcast two or more times. What we found was that a considerable portion of all the network traffic was simply redundant packets. While these packets are necessary for this algorithm to be reliable, if a more efficient algorithm were used, network traffic would decrease by 25%.

As a quick side note, the reason why the first column is zero is that when the Road Train comprises of just a truck and a single car, the Reliable Broadcast Algorithm makes sure you do not send a packet back to the node from where it came.

### Admin Packet Concerns

When we decided to abandon the constant file reading and writing and move to the Admin Packet system, there was some concern that these packets would congest the network and take up valuable bandwidth. However, after some testing, we found that the Admin Packets make up a relatively small portion of the network traffic. These packets are never forwarded through the Reliable Broadcast Algorithm and are therefore not contributing to any detrimental overhead. Compared to the volume of traffic sent by the Reliable Broadcast Algorithm when the Road Train contains three or more cars, the congestion caused by Admin Packets is relatively small. Obviously, in a real world situation, you would not have this extra traffic; these packets are simply there to ensure that the road train simulator works properly.

## CONCLUSION

While the Reliable Broadcast Algorithm is easy to implement and lives up to its name by guaranteeing that nearly every packet will reach every corner of the VANET, the amount of packet flooding that ensues causes major performance issues that may outweigh the benefits of its reliability. In a real world situation, if you had enough bandwidth and processing power this may not be a bad algorithm to use, especially if you are responsible for the lives of everyone driving on the highway. However, other algorithms exist that would be able to deliver similar reliability while optimizing bandwidth and overall performance.

## APPENDIX

### Road Train Algorithm

Assumptions:

1. Any node that is sending us a packet is within 100m
2. The truck is always in the RIGHT LANE and in FRONT
3. Truck speed is in random range of **[20, 35]** m/s
4. Cars can start in LEFT or RIGHT LANE
5. Car starting speed is **[25, 35]** m/s
6. Connecting / disconnecting nodes is done in another layer
   1. We will never be < 100m and not connected to all nodes in that radius

**Road Rules (Not in Road Train)**

1. IF I am in the LEFT LANE, check and see if there is anything within 20 meters in the RIGHT LANE
   1. IF CLEAR, move to RIGHT LANE, resume normal speed (if different)
   2. ELSE increase speed by 5 m/s
      1. Make sure I do not go above some MAX SPEED (Starting speed + 5)
2. IF I get a packet indicating a car is less than 20 meters ahead AND is going slower than me AND IS NOT in a PLATOON…
   1. IF I am in RIGHT LANE
      1. Check if anyone is in LEFT Lane within 20m
         1. IF CLEAR, move to LEFT LANE (Case 1 takes over)
         2. ELSE MATCH the Car’s speed ahead of me
            1. NOTE: We match the speed so we stay at 20 meters behind, this should make us keep checking if the Left Lane is clear (Case 2)
   2. ELSE I am in the LEFT LANE
      1. MATCH the car’s speed ahead of me
      2. Wait until the lane is clear then case 1 should take over

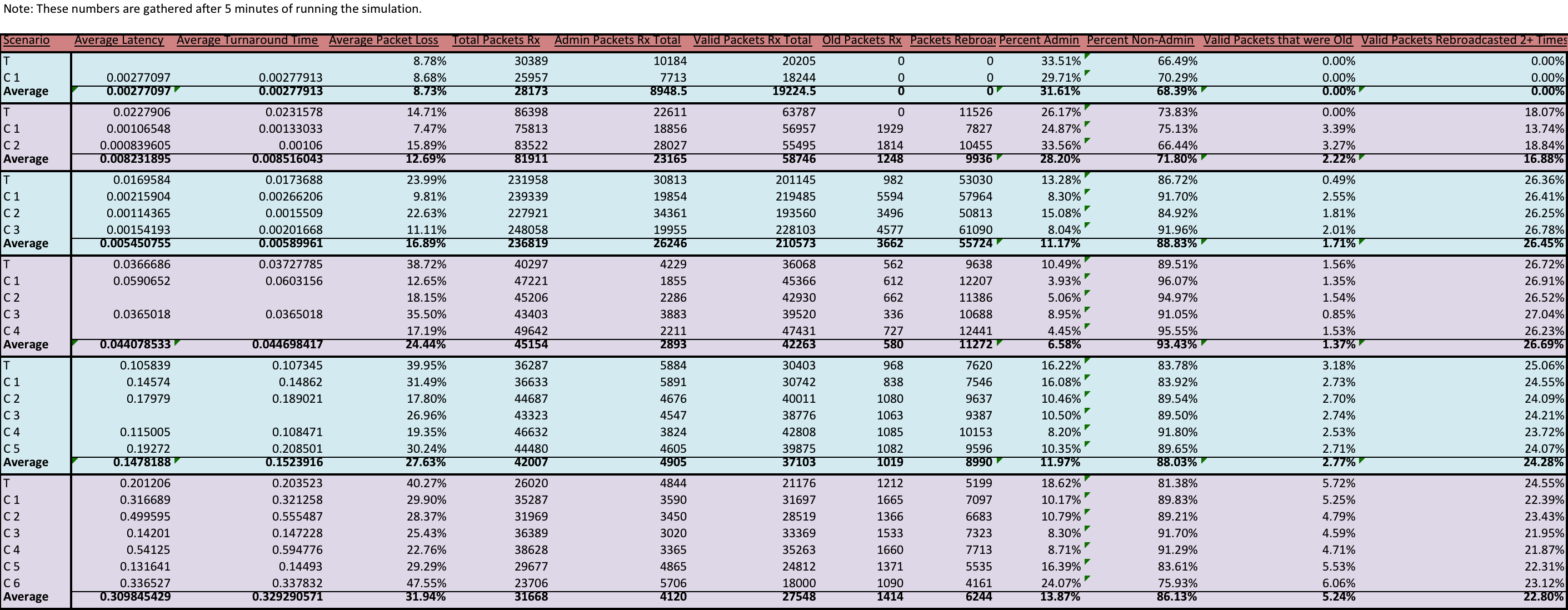
**Road Train Rules - Truck**

1. [TRUCK]: I receive a REQUEST to JOIN the Road Train
   1. Check my variable to see if the Road Train is OPEN
      1. IF OPEN (No other cars are in the process of joining)
         1. Send the car the link number he is (used to calc. distance)
         2. Set Road Train to CLOSED
         3. Wait for ALL CLEAR Packet from Car
      2. ELSE CLOSED (A car is joining the Road Train)
         1. Send the car a packet saying, “TRY AGAIN”
2. [TRUCK]: I receive an ALL CLEAR packet from a car
   1. Set Road Train to OPEN, the next car may now join

**Road Train Rules – Car**

1. IF I am in RIGHT LANE and one of my links is a PLATOON\_MEMBER, I send a REQUEST TO JOIN to PLATOON LEADER
   1. Wait for Response
      1. IF Road Train is OPEN (Received my platoon number)
         1. Increase speed until I am the correct distance away from TRUCK, then match Platoon speed
         2. Send ALL CLEAR PACKET
      2. ELSE MAINTAIN SPEED and keep sending REQUEST PACKETS until OPEN
2. IF I am in LEFT LANE and join a network with a PLATOON
   1. Slow down until I can safely move back to Right Lane
      1. Resume Speed case 1 should take over

**B: Experiment Data**

After we added the fourth car, we had to increase the period of status packets form 10 milliseconds to 100 milliseconds. That explains why those scenarios do not have as many total packets. However, the proportions are still relevant.