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 a Vehicular Ad-hoc Network (VANET). VANET was developed with the intent of making transportation safe and convenient. Car companies of the future may be mandated to install VANET systems in all their vehicles due to the potential safety and economic advantages of such a system. 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 constantly and reliably be transmitted to and from every single car, the bandwidth is limited, speed is of the essence, and everything must be reliable enough so that people would be willing to literally 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
- Timer thread
- Main thread

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

After some initial set up, the main thread goes into an infinite loop that constantly checks the packet buffer. Whenever there is a packet in the buffer, it retrieves it from the front of the queue and processes it. When the packet is being processed, it runs through the Reliable Broadcast Algorithm and will be forwarded as necessary. These packets are also used for updating other node's positions and determining if this node needs to slow down, change lanes, join a platoon, etc.

The timer thread waits for a small 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. This allows us to constantly process packets in the main thread and only send status packets at regular intervals.

DESIGN

The following are some design details on how the program is implemented.

Simulation set-up

When you start the program, it will first check if a Config file exists. If the file does not exist, the program will create a Config file and in turn will know that it is the first node to join and is therefore the truck. If the file does exist, the node is set as a car and reads in the other node information from the Config file. Once the node reads the Config 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 for a status packet to be sent, each node prepares a status packet with its current location in it. Then the current node will 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, however, the packet 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 Config 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 the current locations and connections of every node to Output.txt. This way we can view the current locations of every car at different times throughout 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 discussed in detail in the project specifications, we will not discus 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 algorithm 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 anyone else; we call this the "Road Rules". And two, to enable and maintain platoon formation behind a lead truck, called "Road Train Rules".

The "Road Rules" are as follows. Cars always want to be in the right lane, while 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. If the left lane is clear they will move over and attempt to pass the slower car in the right lane, if 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 case is true, the truck cannot 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 Appendix A.

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 to severe. To resolve this issue, we moved away from our reliance on file reading and writing to a more reliable "Admin Packet" system discussed earlier. This "Admin Packet" system allows us to only have to write to the Config file when a new node first joins. Other than that, the truck does all the output file writing.

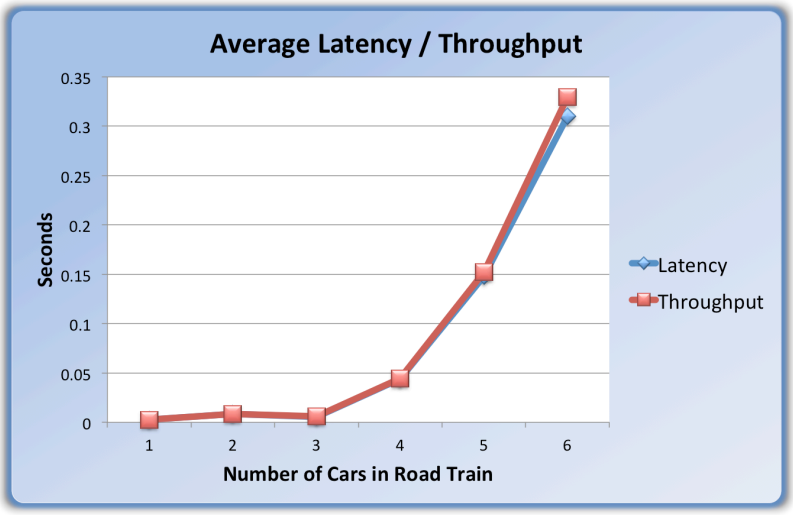
Secondly, we are having issues with timing. We have a function that returns the current time, however, we are sometimes seeing rare cases when it computes a negative difference in time from some previous timestamp. This caused us to have to throw out some of out response time and turnaround time data, 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, the program just couldn't handle all the network traffic while still be responsive enough for the simulation to run smoothly.

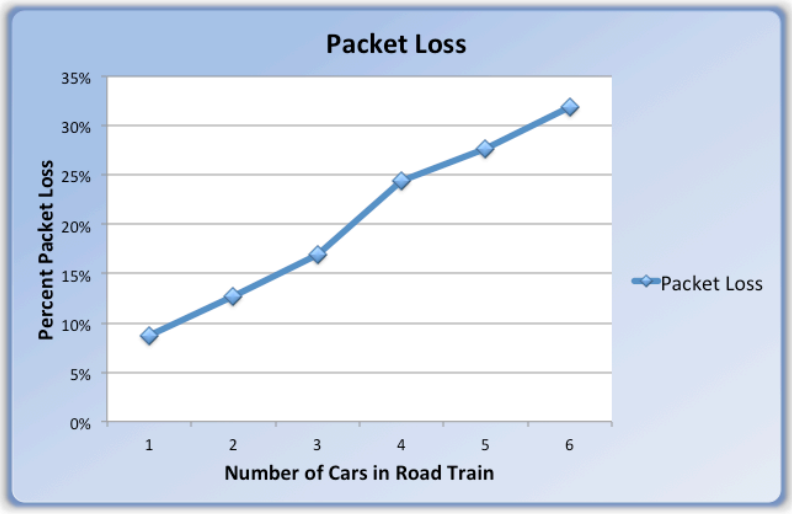
All our experiment data can be found in Appendix B and in the Excel File, which was submitted with this report.

Latency and Throughput



One requirement of any VANET system is that it needs to be fast. If a car breaks unexpectedly ahead of you and your car needs to make an emergency stop, every fraction of a second counts. 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, then 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 drastically. One interesting note was that once the packet was received, it was processed almost immediately. So the bottleneck was not with the processing time, but with the packet buffer just filling up too quickly.

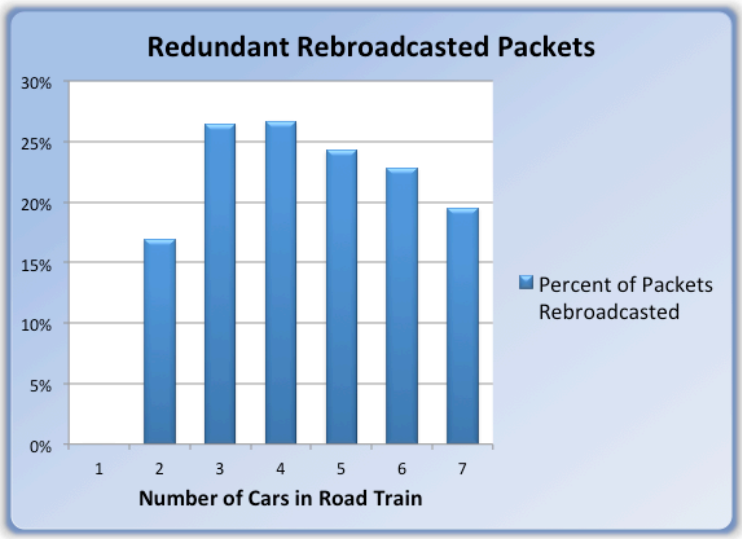
Packet Loss



As with any wireless system, packet loss is always a constant concern. In our experiments we found that packet loss also increased with the number of cars in the road train. However, I do not think this graph explains the whole situation. It appears that the packet loss would increase linearly with the number of cars in the road train. However, I think you would see this graph plateau after 6 or 7 cars. This would be caused by the fact that each node can only transmit 100 meters. So once you have 7 + care in the platoon, you are only really in range of the same amount of cars continuously. So your packet loss would not increase but stay constant. The reason why Road Trains with only 1 or 2 cars have such a low packet loss, is because 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 a greater chance of successfully transmitting to their neighbors.

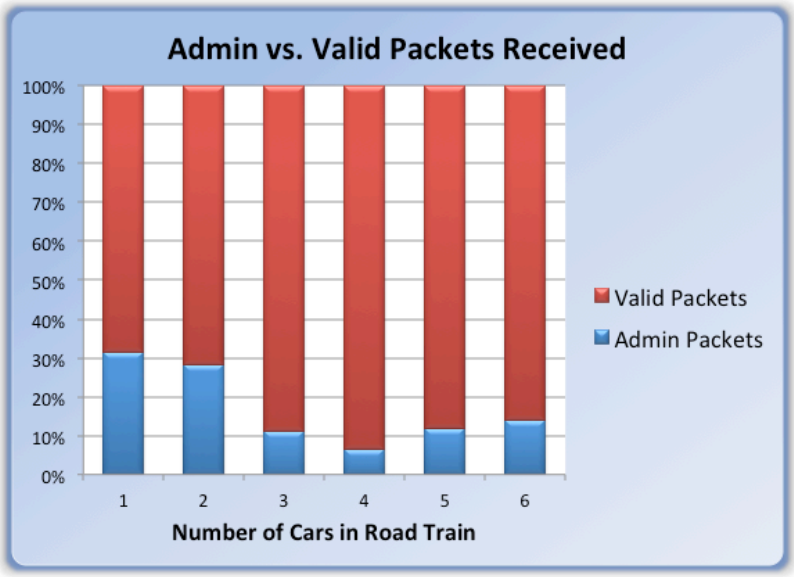
Another interesting observation regarding packet loss is that the lead truck (and the rear car in a long platoon) almost always have the greatest packet loss. This is caused by the fact that the nodes on the end can only transmit to half of their effective area. The truck is only in range of the cars 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 will 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. With that being said, there are a lot of redundant packets being sent around the network. To get an idea of how much of the network traffic is redundant, we kept track of how many packet were rebroadcasted two or more times, so this graph does not take into account the first rebroadcast. 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, you could instantly decrease your network traffic by 25%.

As a quick side note, the reason why the first column is zero is because 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 this "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 not forwarded through the Reliable Broadcast Algorithm and are therefore not contributing to any detrimental overhead. This is especially the case when you have three or more cars in the Road Train and there is already so much other traffic being created from the Reliable Broadcast Algorithm. 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 almost guaranteeing packets 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, I am sure there are other algorithms out there that would be able to deliver similar reliability while optimizing bandwidth and overall performance.

APPENDIX

A: 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
    a. 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
    1.1. IF CLEAR, move to RIGHT LANE, resume normal speed (if different)
    1.2. ELSE increase speed by 5 m/s
        1.2.1. 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…
    2.1. IF I am in RIGHT LANE
        2.1.1. Check if anyone is in LEFT Lane within 20m
            2.1.1.1. IF CLEAR, move to LEFT LANE (Case 1 takes over)
            2.1.1.2. ELSE MATCH the Car's speed ahead of me
                2.1.1.2.1. 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)
    2.2. ELSE I am in the LEFT LANE
        2.2.1. MATCH the car's speed ahead of me
        2.2.2. 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 RoadTrain
    1.1. Check my variable to see if the RoadTrain is OPEN
        1.1.1. IF OPEN (No other cars are in the process of joining)
            1.1.1.1. Send the car the link number he is (used to calc distance)
            1.1.1.2. Set RoadTrain to CLOSED
            1.1.1.3. Wait for ALL CLEAR Packet from Car
        1.1.2. ELSE CLOSED (A car is joining the Road Train)
            1.1.2.1. Send the car a packet saying, "TRY AGAIN"
2. [TRUCK]: I receive an ALL CLEAR packet from a car
    2.1. Set RoadTrain 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
    1.1. Wait for Response
        1.1.1. IF RoadTrain is OPEN (Received my platoon number)

1.1.1.1. Increase speed until I am the correct distance away from TRUCK, then match Platoon speed

1.1.1.2. Send ALL CLEAR PACKET

1.1.2. ELSE MAINTAIN SPEED and keep sending REQUEST PACKETS until OPEN

2. IF I am in LEFT LANE and join a network with a PLATOON

2.1. Slow down until I can safely move back to Right Lane

2.1.1. 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.

Note: These numbers are gathered after 5 minutes of running the simulation.

Scenario	Average Latency	Average Turnaround Time	Average Packet Loss	Total Packets Rx	Admin Packets Rx Total	Valid Packets Rx Total	Old Packets Rx	Packets Rebroa	Percent Admin	Percent Non-Admin	Valid Packets that were Old	Valid Packets Rebroadcasted 2+ Times
T			8.78%	30389	10184	20205	0	0	33.51%	66.49%	0.00%	0.00%
C 1	0.00277097	0.00277913	8.68%	25957	7713	18244	0	0	29.71%	70.29%	0.00%	0.00%
Average	**0.00277097**	**0.00277913**	**8.73%**	**28173**	**8948.5**	**19224.5**	**0**	**0**	**31.61%**	**68.39%**	**0.00%**	**0.00%**
T	0.0227906	0.0231578	14.71%	86398	22611	63787	0	11526	26.17%	73.83%	0.00%	18.07%
C 1	0.00106548	0.00133033	7.47%	75813	18856	56957	1929	7827	24.87%	75.13%	3.39%	13.74%
C 2	0.000839605	0.00106	15.89%	83522	28027	55495	1814	10455	33.56%	66.44%	3.27%	18.84%
Average	**0.008231895**	**0.008516043**	**12.69%**	**81911**	**23165**	**58746**	**1248**	**9936**	**28.20%**	**71.80%**	**2.22%**	**16.88%**
T	0.0169584	0.0173688	23.99%	231958	30813	201145	982	53030	13.28%	86.72%	0.49%	26.36%
C 1	0.00215904	0.00266206	9.81%	239339	19854	219485	5594	57964	8.30%	91.70%	2.55%	26.41%
C 2	0.00114365	0.0015509	22.63%	227921	34361	193560	3496	50813	15.08%	84.92%	1.81%	26.25%
C 3	0.00154193	0.00201668	11.11%	248058	19955	228103	4577	61090	8.04%	91.96%	2.01%	26.78%
Average	**0.005450755**	**0.00589961**	**16.89%**	**236819**	**26246**	**210573**	**3662**	**55724**	**11.17%**	**88.83%**	**1.71%**	**26.45%**
T	0.0366686	0.03727785	38.72%	40297	4229	36068	562	9638	10.49%	89.51%	1.56%	26.72%
C 1	0.0590652	0.0603156	12.65%	47221	1855	45366	612	12207	3.93%	96.07%	1.35%	26.91%
C 2			18.15%	45206	2286	42930	662	11386	5.06%	94.97%	1.54%	26.52%
C 3	0.0365018	0.0365018	35.50%	43403	3883	39520	336	10688	8.95%	91.05%	0.85%	27.04%
C 4			17.19%	49642	2211	47431	727	12441	4.45%	95.55%	1.53%	26.23%
Average	**0.044078533**	**0.044698417**	**24.44%**	**45154**	**2893**	**42263**	**580**	**11272**	**6.58%**	**93.43%**	**1.37%**	**26.69%**
T	0.105839	0.107345	39.95%	36287	5884	30403	968	7620	16.22%	83.78%	3.18%	25.06%
C 1	0.14574	0.14862	31.49%	36633	5891	30742	838	7546	16.08%	83.92%	2.73%	24.55%
C 2	0.17979	0.189021	17.80%	44687	4676	40011	1080	9637	10.46%	89.54%	2.70%	24.09%
C 3			26.96%	43323	4547	38776	1063	9387	10.50%	89.50%	2.74%	24.21%
C 4	0.115005	0.108471	19.35%	46632	3824	42808	1085	10153	8.20%	91.80%	2.53%	23.72%
C 5	0.19272	0.208501	30.24%	44480	4605	39875	1082	9596	10.35%	89.65%	2.71%	24.07%
Average	**0.1478188**	**0.1523916**	**27.63%**	**42007**	**4905**	**37103**	**1019**	**8990**	**11.97%**	**88.03%**	**2.77%**	**24.28%**
T	0.201206	0.203523	40.27%	26020	4844	21176	1212	5199	18.62%	81.38%	5.72%	24.55%
C 1	0.316689	0.321258	29.90%	35287	3590	31697	1665	7097	10.17%	89.83%	5.25%	22.39%
C 2	0.499595	0.555487	28.37%	31969	3450	28519	1366	6683	10.79%	89.21%	4.79%	23.43%
C 3	0.14201	0.147228	25.43%	36389	3020	33369	1533	7323	8.30%	91.70%	4.59%	21.95%
C 4	0.54125	0.594776	22.76%	38628	3365	35263	1660	7713	8.71%	91.29%	4.71%	21.87%
C 5	0.131641	0.14493	29.29%	29677	4865	24812	1371	5535	16.39%	83.61%	5.53%	22.31%
C 6	0.336527	0.337832	47.55%	23706	5706	18000	1090	4161	24.07%	75.93%	6.06%	23.12%
Average	**0.309845429**	**0.329290571**	**31.94%**	**31668**	**4120**	**27548**	**1414**	**6244**	**13.87%**	**86.13%**	**5.24%**	**22.80%**