Software Define Network for Traffic Management

Juliet Villanueva Casey Garrett and Aaron Dartt
College of Engineering, University of Nevada, Reno
1664 N Virginia St, Reno, NV 89557

Abstract—**The rapid growth of technology is leading cities to become more intelligent providing useful data gathering from all aspects of infrastructure which is then managed by software defined networks(SDN). For this paper, a traffic managed SDN is used to create the optimal path for network connected autonomous cars to drive from their source location to their end destination. The simulation was created with C++ with assumptions made more clear later in the paper. The paper focuses on the use of Dijkstra's algorithm to find the best path. The system is proactive and reactive in that while a large set of cars are in the graph (road way) each will be taking up edge units causing the algorithm to change path choices throughout the simulation. With the current simulation results this algorithm appears invariant to the amount of cars present in the graph if they are added individually.**

Keywords- Proactive; Software defined network(SDN); throughput; Reactive;

I. INTRODUCTION

Driverless cars are progressing in day to day life. Most recently, the widely adopted Tesla technology, with limited driverless capabilities but it does control the car and keep its passengers safe. Future research and development can lead to all cars becoming driverless and with more time become fully autonomous. A logical step to fulfill the goal of autonomy with cars would be a software defined network (SND). What this technology would do is gather information from all cars on the network, and as data flows the best possible routing directions are given. The best paths would be calculated by Dijkstra's algorithm via a main database which the cars will constantly push information to, and in turn receive directions to get from point A to point B. The simulation centers on only cars within a finite graph controlled roadway. Additional factors could be adapted with further research, such as weather conditions, road conditions, school zone speed changes and much more. Once more systems are controlled via SDN's each can communicate with each other and provide optimal outcomes for vast areas within cities.

II. Dijkstra's & the Database

In the simulation the database is the only component that makes complex computations. The cars simply push information and pull from the database. Inside the database is the main graph, which for testing purposes is an N by N bidirectional matrix filled with randomly generated edge values. The edge values are arbitrary units of time. For example, in figure 1 the edge from vertex (5) to (1) is three time units. The units of time are relative and simply convey the different units of time throughout the graph. The database also holds a vectors of cars. Each data type of car has a destination, source, currentpath and time variables to keep track of the car. The time variables hold values such as; initTime, timeStart, timeToEnd, and totalTime. These variables are being used while a car is in the graph and is driving from one node to another. Each time the car moves to another node, previous edges values are decremented. The additional variables such as currentpath hold the route the database has given to the car. As more cars enter the graph and calculated paths are generated, edge values are increased to simulate where a car is or will be using said edge path. Since the simulation is proactive, car A will have the optimal path for the current graph G, then car B will get the optimal path with a newly updated graph G + E[]. E[] being an array of the edge values which will increase by one because car A is using the path. The simulation will also be reactive in that while a car has traveled an edge the value of the edge is decreased. The value lowers because the car is done and will no longer need that edge path in the future. As more cars take commonly calculated paths, edges become congested. These congested paths indicate a theoretical increase in time to traverse the edge. As the graph is updated with more cars and larger edges values, the throughput of the originally calculated shortest path changes. What may have taken 27 edge units to traverse, will now take 2540 edge units as more cars cause congestion.

III. Assumptions for Simulation

As was stated in the introduction many factors can be implemented into the simulation. However, due to the constraint of time, assumptions were made in regards to the simulation. Firstly, the simulation is for a network in which all vertices and cars are reachable via the network. The focus is on changing values within a fixed graph. Secondly, the network is under perfect conditions, meaning no loss of connection with the database can occur. Finally, there are a static number of cars which travel through the graph during the simulation. In the data section of this document smaller values are used to help understand what is happening in the simulation. From these assumption, the program best simulates rush hour for a set of cars all entering at very similar time frames. The focus of the simulation is to gather data on thousands of cars driving at once and be able to continue directing preceding cars which enter the graph, resulting in best possible paths to be re-routed. The results of the simulation and the breakdown of the data is in the follow section.

IV. Results and Data Collection

The construction of data was amounted using five different test cases. Each test case once executed will generate random edge values and cars which will generate random source and destination locations. The starting graph is written out in figure 1. The base graph in figure 1 has a yellow line, which represents the shortest path from vertex (0) to vertex (7) taking 28 time units. After the program runs with a random number of cars the optimal path changes. The new shortest path from (0), to (7) is the blue line. The simulation being bombarded with additional cars resulted in a re-routed path. Similar to actually driving in rush hour, cars going from vertex (0) to (7) can be given the yellow path if they enter the graph/roadway earlier, while cars in the thick of rush hour may be re-routed for optimal time.

The simulation has two different types of algorithms used. One algorithm, Reserved Path, increments each edge along the path discovered by Dijkstra's algorithm. The second algorithm, Non Reserved Path, only increments the edge that the car is currently on. Each algorithm will add the car to the graph and then after enough time steps have passed, i.e. the weight of the current edge, the car will see if a new better path is available and then update its path based on the algorithm being used.

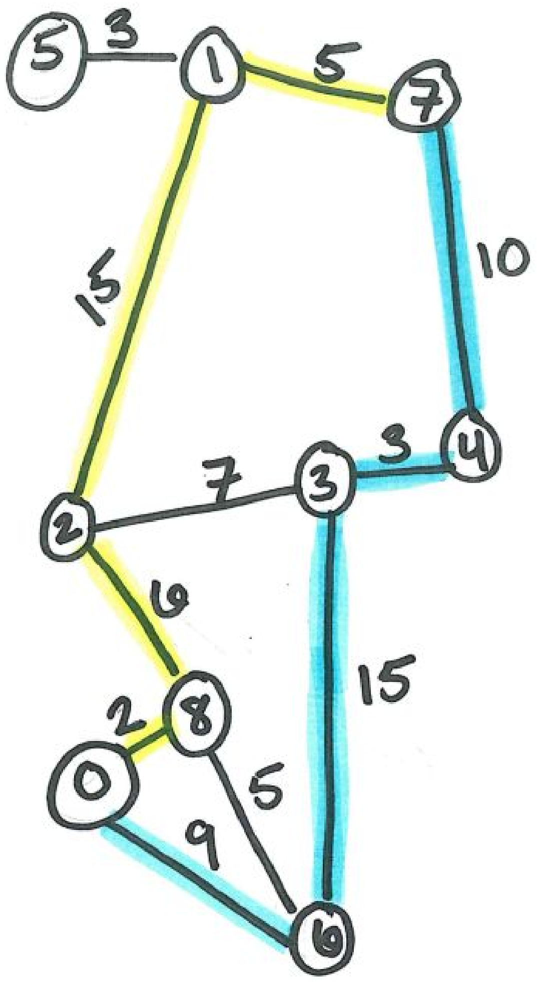
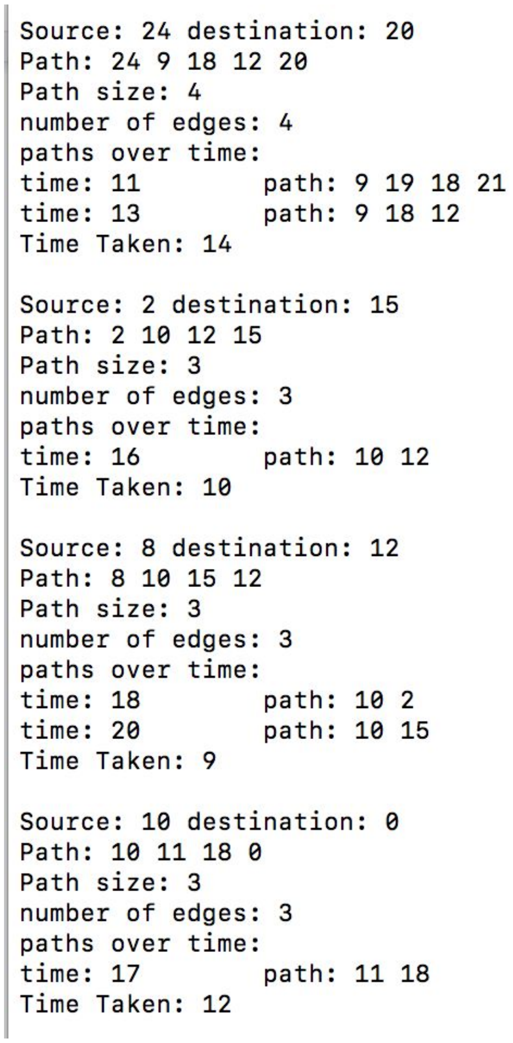


Fig. 1 A path that was changed during a run of the program.

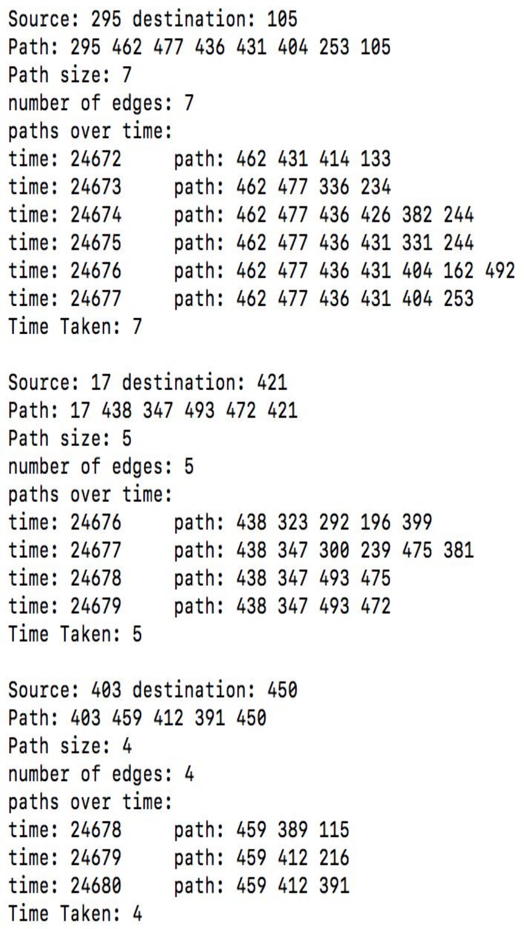
The results of the two algorithms was interesting. The Reserved Path algorithm took significantly longer for a car to traverse the graph on average 21.5 time steps for the 25x25 graph. Some cars would change paths during the traversal as shown in Fig. 2. The results of the Non Reserved Path algorithm on the same graph with the same amount of cars had a much faster average traversal rate of around 12.57 time steps. The algorithm also changed paths much more frequently than the Reserved Path algorithm as shown in Fig 3.



```
Source: 24 destination: 20
Path: 24 9 18 12 20
Path size: 4
number of edges: 4
paths over time:
time: 11        path: 9 19 18 21
time: 13        path: 9 18 12
Time Taken: 14

Source: 2 destination: 15
Path: 2 10 12 15
Path size: 3
number of edges: 3
paths over time:
time: 16        path: 10 12
Time Taken: 10

Source: 8 destination: 12
Path: 8 10 15 12
Path size: 3
number of edges: 3
paths over time:
time: 18        path: 10 2
time: 20        path: 10 15
Time Taken: 9

Source: 10 destination: 0
Path: 10 11 18 0
Path size: 3
number of edges: 3
paths over time:
time: 17        path: 11 18
Time Taken: 12
```

Fig. 2 The output from the Reserved Paths algorithm that change over time. Note that the paths change less frequently than that of the Non Reserved Paths algorithm.

As the graph size increased the amount of time steps decreased for both algorithms. The Reserved Path Algorithm with 10,000 cars took an average of 21.5 time steps on a 25x25 graph while for the same amount of cars it only took an average of 10.784 time steps on the much larger 500x500 graph. The Non Reserved Path algorithm had similar results, with 10,000 cars it took an average of 12.57 time steps for a car to finish the traversal.



```
Source: 295 destination: 105
Path: 295 462 477 436 431 404 253 105
Path size: 7
number of edges: 7
paths over time:
time: 24672     path: 462 431 414 133
time: 24673     path: 462 477 336 234
time: 24674     path: 462 477 436 426 382 244
time: 24675     path: 462 477 436 431 331 244
time: 24676     path: 462 477 436 431 404 162 492
time: 24677     path: 462 477 436 431 404 253
Time Taken: 7

Source: 17 destination: 421
Path: 17 438 347 493 472 421
Path size: 5
number of edges: 5
paths over time:
time: 24676     path: 438 323 292 196 399
time: 24677     path: 438 347 300 239 475 381
time: 24678     path: 438 347 493 475
time: 24679     path: 438 347 493 472
Time Taken: 5

Source: 403 destination: 450
Path: 403 459 412 391 450
Path size: 4
number of edges: 4
paths over time:
time: 24678     path: 459 389 115
time: 24679     path: 459 412 216
time: 24680     path: 459 412 391
Time Taken: 4
```

Fig. 3 The output from the Non Reserved Paths algorithm that change over time. Note that the paths change more frequently than that of the Reserved Paths algorithm.

There was almost no change in the average amount of time taken when the number of cars on the graph was increased. With the Reserved Path algorithm, on a 25x25 graph with 10,000 cars the average was 21.5 time steps with 50,000 cars on the same graph the average time for a car to traverse the graph was 21.52 time steps. The same algorithm on a graph of 500x500 graph had an average of 10.784 time steps for a car to traverse the graph with 10,000 cars and an average of 10.772 time steps with 50,000 cars. The result was very similar with respect to the Non Reserved algorithm which had an average of 12.57 time steps with 10,000 cars and an average of 12.61 time steps with 50,000 cars on a 25x25 graph. With the same algorithm on a 500x500 graph, the average time for a car to traverse the graph was 4.781 and 4.93512 for 10,000 and 50,000 cars respectively.

V. CONCLUSION

The results from this experiment had some very interesting outcomes that were not immediately obvious. The fact that no matter how many cars were added to a graph, the average amount of time steps needed to finish did not change and no noticeable increases were observed at the sizes tested. Although, there was an obvious difference observed with respect to the two different algorithms used. The Non Reserved Path algorithm took less time steps for a car to traverse the same graph as the Reserved Path algorithm. This result was expected, as the graph was not populated with a full path in the Non Reserved Path algorithm, allowing cars to find faster paths more frequently. The best algorithm, by every metric, is the Non Reserved Path algorithm. Unfortunately the computers used in this experiment were not powerful enough to test higher volumes of cars limiting the test to only 50,000 cars.

In the future it would be nice to introduce more variables in the cars. This would allow for edge weights to be custom per car depending on the size, speed, and lane restrictions. It would also behove future experimenters to consider using parallel computing, harnessing the power that can only come in the form of High-Performance Computing Cluster or HPCC. Utilizing the HPCC would allow for larger graphs and higher traffic volumes, it would also increase the speed that algorithm can finish each time step.