Performance Testing and Validation of a Traffic Simulator

Rebecca Bari Buchheit
B.S. Civil Engineering, Carnegie Mellon University

submitted in partial fulfillment of the requirements
for the degree of
Master of Science in Computer Aided Engineering and Management

Department of Civil and Environmental Engineering
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

August 1998

Abstract

DynaMIT is a real-time system that can be used to provide travel guidance and traffic condition information to drivers. The traffic simulation module of DynaMIT is a macroscopic model that simulates the movement of large groups of vehicles on a traffic network. Because DynaMIT is used to give real-time guidance to travelers, the run time performance and the correctness of the system is critical. To this end, a series of tests were devised to evaluate the accuracy and performance of the traffic simulator.

A homogeneous, square grid network and a model of a real traffic network in Boston, Massachusetts were both used to evaluate the simulator. The primary function of the grid network was to test the simulator's performance over a wide variety of network sizes and traffic density patterns. The real traffic network was used to evaluate both the run time performance and the accuracy of the simulation.

Acknowledgments

I am indebted to my advisor Dr. Haris Koutsopoulos for his guidance and patience throughout this project. His meticulous attention to detail and his understanding of DynaMIT were instrumental to the work that I have done.

I would also like to thank Sridhar Rajagopal, Michel Bierlaire, and Michael Heiler for all of their help and support. Their timely assistance helped to keep the project on schedule and saved me from endless hours of frustration.

Rebecca Buchheit

Contents

List of Figures

List of Tables

1 Introduction

The Dynamic Network Assignment for the Management of Information to Travelers (DynaMIT) system is a real-time system used to provide travel guidance and current traffic conditions to drivers. The system is currently under joint development by the Massachusetts Institute of Technology (MIT), Carnegie Mellon University (CMU) and The Analytical Sciences Corporation (TASC).

The objective of this thesis is to demonstrate that the traffic simulation module of DynaMIT provides correct traffic simulation and is capable of real-time operation. Therefore, this thesis is divided into two main sections: a section detailing the performance tests used to characterize simulation time and a section detailing the tests used to validate the results of the simulator.

The traffic simulator (also referred to as the supply module) is a major component of DynaMIT. It simulates the movement of vehicles (packets) on a network specified by the network topology and path topology components of DynaMIT. The simulator is a macroscopic model; its primary functionality is to describe the movement of large groups of packets at an aggregate level. The DynaMIT origin-destination (OD) estimation module generates a description of packets that wish to enter the network. The complete description of these packets, their origin, destination, and physical parameters, is called the packet demand and is used as an input to the supply module. The network and path topologies remain static for a given network; the packet demand may change for each run of the simulation.

The supply module is used to calculate the current state of the network and to predict the future state of the network. The result of each simulation is a detailed description of the movement of packets on the network as well as an aggregate description of the overall state of the network. These results are used by the guidance generation module to give travel advice based on the estimated future state of the network and by the OD module to update its model of the packet demand.

For the purposes of testing, the interfaces between the supply module and the other components of DynaMIT have been severed. The interaction between the network and path topology components and the supply module is simulated with temporary debugging code. The input that these components would provide to the supply module has been generated in two ways:

with a grid generation program and with text files containing a fixed topology describing a real network. The interaction between the OD estimation and guidance generation modules and the supply module is also simulated with text files and temporary code.

1.1 Network Components and Output Files

The input to the supply simulator consists of a network topology and a description of packet demand for the network. The network topology has six major components: nodes, loaders, links, segments, lane groups, and lanes. Sensors are used to gather traffic data during the simulation. A path topology file describes how the links in the network are connected.

The supply simulator outputs five different files: travel times for packets, segment densities, segment flow, packet speed on a segment, and queue lengths for lane groups. The flow and speed data are collected by the sensors. The simulation algorithm calculates the remaining data directly. Each file is updated with new data at the end of each update interval.

1.1.1 Nodes and Loaders

A node represents an intersection in a traffic network; each node is connected to a downstream and/or upstream link. Loaders are associated with a node. Traffic is loaded onto the network by upstream links connected to a loader and removed from the network by downstream links connected to a loader.

Each node is described by an identification number, identification numbers of all of its connected upstream links, and identification numbers of all of its connected downstream links. Loaders are specified by the identification number of the node to which they are connected, an output flow (vehicles per second) and an input flow (vehicles per second).

1.1.2 Links, Segments, Lane Groups, and Lanes

Links are one-way connections between nodes in the network. Each link consists of one or more segments, which are connected end to end. Segments are used to differentiate between parts of a link where the number of lanes and

the traffic parameters may vary. Each segment contains one or more lane groups. Lane groups are used to route traffic to the next link. For example, a segment might have three lane groups: one for traffic turning left, one for traffic going straight, and a third for traffic turning right. Each lane group can contain one or more lanes. Lanes are a representation of actual lanes in a real traffic network; packets travel in lanes.

Each link is described by an identification number and a list of segments that the link contains. Segments are defined by a parent link identification number and their position within the parent link. Each segment also has a length (meters), a free-flow speed (meters per second), a jam density (vehicles per meter), an alpha parameter, a beta parameter, a list of lane groups that the segment contains, and a list of sensors that are located on the segment. The jam density and alpha and beta parameters are used to calculate the speed at which vehicles can travel on a segment, given the density of the segment. Sensors are specified by the segment on which they reside and their location on the segment.

The lengths associated with links and segments are measured starting from the downstream end of the link or segment. Packets enter links and segments at their upstream ends and move towards their downstream ends. Thus, the position of a packet at the upstream end of a link is equal to the length of the link, while the position of a packet at the downstream end of a link is zero.

Each lane group is specified by its parent segment identification number, output capacity (vehicles per second), a list of lanes contained in the lane group, and a list of possible downstream links. Finally, each lane is described by its parent lane group and a list of possible downstream links.

1.1.3 Paths

The path topology component contains a list of viable paths that packets can use when traveling through the network. The path topology consists of two sets of path and link identification number pairs. The first path and link pair is the current state of the packet; the second path and link describes the next state the packet will be in once it leaves the current state.

1.1.4 Packet Demand

The packet demand component describes all of the packets that will be loaded onto the traffic network. Each packet has an origin loader where it will be loaded onto the network, a destination loader where it will be removed from the network, and an estimated departure time. The current link and path of the packet, the next link and path of the packet, the current segment that contains the packet, and the position of the packet on the current link describe the starting state of the packet. Additionally, the physical length of the packet and the number of vehicles in the packet are parameters included in the description of each packet.

1.1.5 Output Files

After each update interval ends, the simulator collects any information gathered by the sensors. Whenever a packet crosses a sensor, the current simulation time and the speed of the packet are recorded. The flow information file contains the sensor identification number, the packet identification number, and the time (seconds) when the packet crossed the sensor. The speed information file contains the sensor identification number, the packet identification number, and the speed (meters per second) at which the packet was traveling when it crossed the sensor.

The simulator also gathers information from segments and lanes at the end of each update interval. From the segments the simulator reports the density of packets on each segment, and from the lanes the simulator reports the queue length in each lane. The densities are recorded in the density information file, which contains the segment identification number and the segment density (vehicles per meter) at the end of the update interval. The queue lengths are recorded in the queue length information file, which contains the lane identification number and the queue length in that lane at the end of the update interval.

The final output file is the travel time information file. Each time a packet advances to a new link, the time that it took to traverse the previous link is recorded. At the end of each update interval, this information is written to the travel time information file, which includes the packet identification number, the identification numbers of the previous and current links, the time (seconds) at which the packet arrived at the previous link, and the

amount of time (seconds) that the packet took to travel the previous link.

1.2 Supply Simulator Module Algorithm

In order to begin a supply simulation, the `initializeSupplyModule` method must be called. This method generates a physical description of the network from the path and network topology components. It also generates a list of packets from the packet demand component. Once the simulation is initialized, the `simulateTraffic` method is used to run the actual simulation. At this point, the user specifies the length of the simulation (seconds), the update interval (seconds), and the advance interval (seconds). The `simulateTraffic` method calls the `advanceTraffic` method for each advance phase and the `updateTraffic` method for each update phase.

1.2.1 `advanceTraffic` Method

The `advanceTraffic` method begins by initializing the queue length in each lane in the network. Then it gets a list of nodes, sorted by the order in which they should be processed by the simulation. For each node in the sorted list, a list of upstream packets is generated and sorted by estimating when the packet will pass through the node. For each of these packets in the sorted list, the `advancePacket` method is called to move the packet in the network. Once all of these packets have been moved, the upstream segments of the node are marked as being processed. When all of the nodes have been processed, the simulation calls the `advanceAllVirtual` method.

The `advanceAllVirtual` method was created to deal with concurrency issues. In a real traffic network, vehicles move concurrently; in the simulation, vehicles are moved sequentially. Since segments can not be processed in the direction of travel for all packets, some downstream segments may be processed before their upstream connections are processed. In order to keep packets located on upstream segments from moving to saturated downstream segments that have already been processed, the idea of a virtual queue was developed. The available length of the virtual queue is set to the input capacity of the downstream segment at the beginning of each update phase. When the upstream segment is processed, packets that wish to move to the downstream segment are placed on its virtual queue, if its capacity permits.

The `advanceAllVirtual` method loads these packets onto the downstream segment. Finally, the `advanceTraffic` method calls the `loadNewPackets` method. This method loads packets that are waiting at the node onto the network, if the downstream segments have sufficient capacity.

1.2.2 `updateTraffic` Method

The `updateTraffic` method updates the state of all of the lane groups, links, loaders, and segments in the network and oversees the addition and removal of incidents from the network. It also calls the `reportFlows`, `reportSpeeds`, `reportQueueLengths`, `reportTravelTimes`, and `reportDensities` methods, which report simulation information to the output files.

2 Test Design

In order to thoroughly test the simulator, we used two different network scenarios: a grid network and the Central Artery and Tunnel (CA-T) network, which is located in Boston, Massachusetts. Each scenario can further be divided into three components: the network topology, the packet demand, and the granularity of the simulation. Variations of each possible scenario and component pair were used for testing.

The first scenario is the grid network, a square network that can easily be generated with a computer program. This flexibility allowed us to generate a wide variety of possible topologies and packet demand rates. Each grid allows the user to specify a link length, loader input and output flows, jam density on segments, alpha and beta (parameters used to determine the speed of packets based on segment density), lane group output capacity, and a grid size of m rows and n columns. The result is a grid with homogeneous components that contains $m * n$ nodes with loaders connected to the edge nodes. The loaders on the west and north edges load packets onto the network; the loaders on the east and south edges unload packets from the network. The resulting topology consists of a set of paths that run west-east and north-south. Each link contains one segment, each segment contains one lane group, and each lane group contains one lane. There is one sensor located on each segment. The user can specify the packet demand rate by setting the number of vehicles that are loaded at each loader on the network per hour. The result is a packet file in which the packets are evenly

divided between the possible west-east and north-south paths.

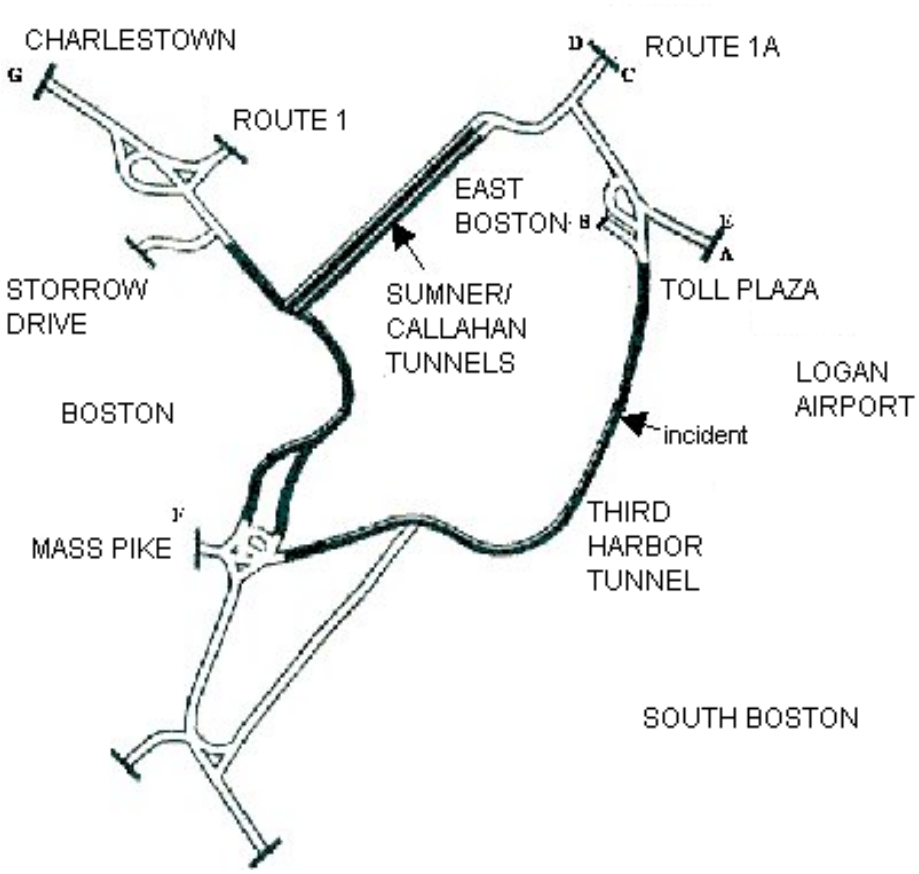

Figure 1: CA-T Network

The second scenario is the CA-T network (see Figure 1). This network represents the connection between Route 1A and Logan International Airport with I-93, Route 1, and the Massachusetts Turnpike. It contains 211 links, 639 segments and lane groups, 1259 lanes, 182 nodes, 64 loaders, and 708 sensors. Since this network represents a real traffic scenario, the topology of the network is fixed. Although the CA-T network contains fewer components than many of the possible grid networks, the added complexity of its connections provides a good test bench. The packet demand file for the CA-T network is generated by an origin-destination module. The resulting

file can be edited by hand to allow the user to vary the packet demand for the simulation. Like the network topology, the packet demand may be smaller than the demand for some of the grid network configurations, but the complexity of the paths that the packets use is a useful addition to the test matrix.

Finally, the simulator itself allows the user to vary the granularity of the simulation. The user specifies the simulation time, advance interval, and update interval. Shorter advance intervals lead to a finer grained simulation. Shorter update intervals lead to a finer grained set of output files.

3 Validation

The first phase of the testing process was the process of validating the correctness of the supply simulator. All of the validation tests were performed on the CA-T network. The regularity of the grid networks make them a poor test bench for validity checks, whereas the complexity of the topology of the CA-T network is well suited for these checks. The best way to validate the simulator would have been to compare the results of the simulation with actual traffic data. Unfortunately, no traffic data exists for the CA-T network at this time so we were forced to develop other methods to check the validity of the simulations. We used two different methods to test the simulator: graphically representing the results and testing the sensitivity of the simulator to different parameters. In the future, it would be useful to cross validate with MITSIM. MITSIM is a microscopic traffic simulator developed at the Massachusetts Institute of Technology, and it has the advantage of using input files very similar to those used by DynaMIT.

3.1 Database Design

The first step of the validation process was developing a database to catalog the input components and output files from different simulation runs. This database allowed us to easily produce files that were formatted correctly for the graphing and mathematical software that was used for the validity tests.

All of the network topology components, output files, and packet information files are included in the database. The `zNetworkFile` is normalized by a perl script to remove redundant data and is split into eight separate files: zLinks,

zSegments, zLaneGroups, zLanes, zNodeUplinks, zNodeDownlinks, zLoaders, and zSensors. Each file will become an entity (table) in the database. Since the members of each entity are unnumbered in the `zNetworkFile`, part of the normalization process is assigning an identification number (starting at 0) to each member of the entity. The normalization process removes the next link identification numbers from the lanes, the child lane identification numbers and next link identification numbers from the lane groups, the child segment identification numbers from the links, and the child lane group identification numbers and sensor identification numbers from the segments. All of this information is stored elsewhere, making it redundant to include in the database again. The normalization process also splits the node information into two tables: one for upstream connecting links and one for downstream connecting links.

The `zPathTopoFile` is a single entity and is imported directly into the database. The `zPacketFile` is also a single entity. It is imported directly into the database, and the database assigns packet identification numbers for each member. We edited the `zPacketFile` by hand to produce files with demands of 10,000 packets and 36,000 packets (in addition to the original file with 18,244 packets); these files are included in the database as separate tables.

The output files from different runs of the simulator are cataloged separately. Thus, for nine different runs, there are nine sets of five output file tables. The speed, flow, and travel time information files can be imported directly into the database. The density and queue length information files must be normalized before they are imported. In order for each member of the density and queue length entities to be unique, a time-step attribute was added to the output files. The time-step attribute is the update interval number when the density and queue length are recorded.

The entity-relationship diagram and the table descriptions (see Figure 2 and Appendix A) show the relationships between the various entities in the database. A primary key (shown in boldface) is the value that makes each member of the entity unique. Some entities may have multiple primary keys. In this case, each unique combination of the primary keys ensures the uniqueness of the member. Some attributes have referential integrity constraints. A referential integrity constraint limits the value of the attribute to a set of values in another attribute. For example, a referential integrity constraint to the attribute zLinks.linkID limits the value of this attribute

Figure 2: Entity-Relationship Diagram

to the set of identification numbers in the attribute linkID, contained in the entity zLinks. Finally, the relationships between entities are shown. All of the relationships in the DynaMIT database are one-to-many relationships. A one-to-many relationship implies that for every one instance of an entity there are zero or more related instances of the second entity. For example, there is a one-to-many relationship between links and segments; one link may contain many segments.

3.2 Database Rationale

Including the path topology and packet demand component data in the database allowed us to link the topology of the network to the output produced by the simulator. For example, queue lengths are expressed in terms

of the lane in which they occur. Flow and speed are expressed in terms of the sensor that records their values. In one of the validity tests, we wanted to check that the queue lengths for certain segments were consistent with the density, flow, and speed on those segments. The database allowed us to associate all of the lanes in a lane group with their parent lane group and then to the segment containing that lane group. It also associated the sensors with the segments that contain them. The results allowed us to compare queue lengths, density, flow and speed across segments without having to manually look up the queue values for the lanes and the flow and speed values for the sensors in those segments.

The inclusion of the packet demand files in the database allowed us to verify that packets were on the correct path and that the loaders were functioning correctly. The packet identification numbers were used to find missing and duplicate packets caused by errors in the simulator.

The database was also useful for screening the output files for physically impossible values. For example, since the packet demand file specifies that all packets are five meters long, it would be physically impossible to have a segment density greater than 200 vehicles per kilometer. The database was able to sort the density output file by the densities, which showed the segments with incorrect densities easily.

3.3 Graphical Representation of Results

Graphical representation was used to verify the consistency between the results from various files. We chose several segments to represent; however, segment 209 and its downstream and upstream neighbors are presented here because of their unique geometry (see Figure 3 and Table 1).

The charts (see Figures 4, 5, and 6) were created from the results of a simulation of one hour with an update interval of 60 seconds and an advance interval of 30 seconds. The density is shown as the instantaneous number of vehicles on the segment at the end of each update interval. The queue length is shown as the instantaneous number of queued cars in all of the lanes in the segment. The flow in and the flow out of each segment are represented in two ways. The number of vehicles entering the segment per minute is shown as the dashed gray line; the number exiting the segment per minute is the dashed black line. The cumulative number of vehicles that

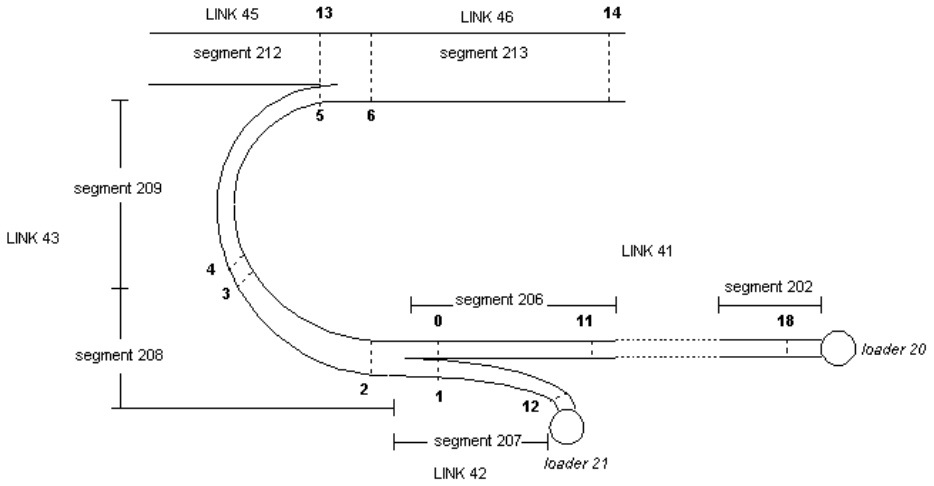

Figure 3: Segment 209 Geometry

Segment	Length (m)	Number of Lanes	Parent Link	Flow In		Flow Out	
				Sensor	Location	Sensor	Location
202	313.05	1	41	18	310.0	-	-
206	282.179	1	41	11	278.0	0	2.5
207	521.858	1	42	12	515.0	1	2.5
208	164.222	2	43	2	160.0	3	2.5
209	703.935	1	43	4	700.0	5	2.5
212	380.345	3	45	-	-	13	2.5
213	742.906	4	46	6	738.0	14	2.5

Table 1: Segment Geometry and Sensor Placement

have entered the segment since the start of the simulation is shown as the solid gray line; the number that have exited is the solid black line.

At each point in the chart, the difference between the cumulative number of vehicles that have entered and exited should be equal to the number of vehicles on the segment (density). Each segment has maximum capacities for density and for flow out of the segment. The maximum density is a function of the segment's length and the number of lanes: $density_{max} = \frac{length_{segment} * lanes_{num}}{length_{packets}}$. The maximum flow out of the seg-

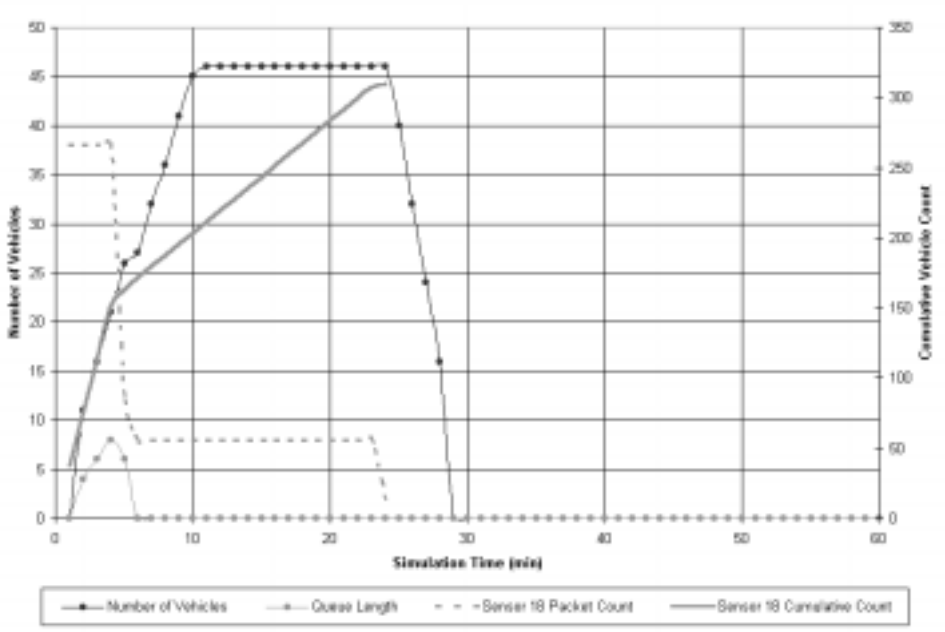

Figure 4: Segment 202 Chart

ment is actually associated with lane groups. However, in the CA-T network, each segment has only one lane group. The maximum flow out is $flow_{max} = flow * lanes_{num}$; in the CA-T network, $flow$ is equal to 0.55 vehicles per second. The flow into the segment is determined by the speed of the packets that are traveling on the segment and the queue length of the lanes in the segment.

Segment 202 is directly attached to a loader, which loads packets exclusively onto segment 202. 148 packets are scheduled to load at zero seconds and 162 packets are scheduled to load at one second into the simulation time. This ensures that the loader has a constant stream of vehicles to release onto the segment. A sensor is located within five meters of the beginning of the segment, near the loader. Note that a small queue begins to form after the first minute and dissipates after the fifth minute. This queue forms because the loader has a higher output capacity than the output capacity of segment 202. Thus, vehicles are loaded onto the segment faster than they can be released. The formation of the queue slows traffic, which the loader

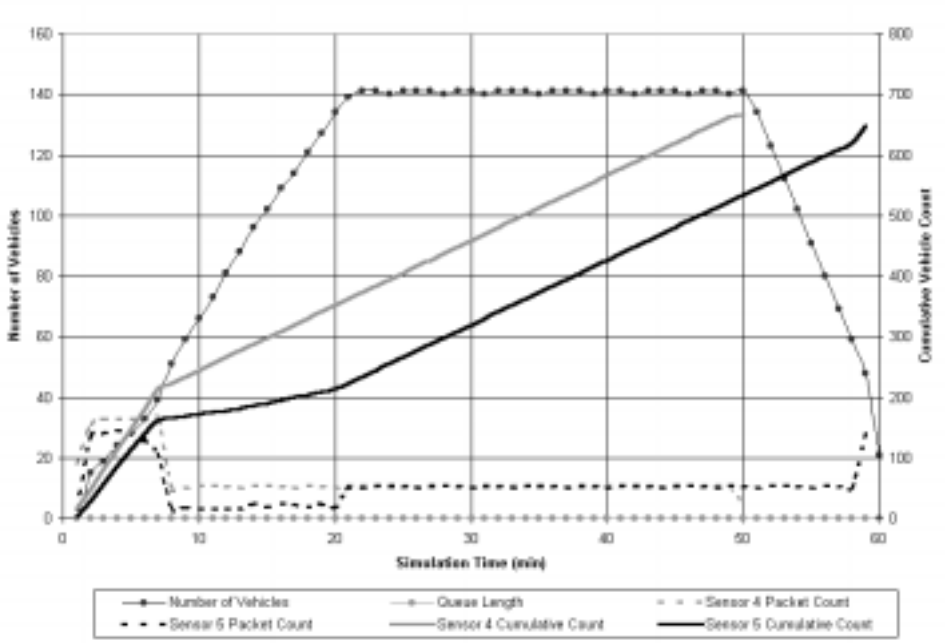

Figure 5: Segment 209 Chart

responds to by decreasing its output. After the tenth minute, the density of the segment is constant until the loader output drops sharply at 23 minutes. During this time, a steady stream of vehicles is loaded at the rate of seven vehicles per minute.

Segment 209 is downstream from segment 202 and from segment 206, which is also attached to a loader. At the beginning of the simulation, segment 209 accepts vehicles at close to its capacity of 33 vehicles per minute. However, as the segment fills to capacity (roughly 140 vehicles), the speed of the vehicles on the segment begins to slow. This retards the rate at which the segment will accept new packets. By the tenth minute of the simulation, segment 209 is accepting about 12 vehicles per minute and continues to do so during the rest of the simulation.

Segment 213 is downstream from segment 209 and from segment 212. The output from segment 212 overwhelms that of segment 209, making the effects of the output of segment 209 difficult to discern. Segment 212 is located

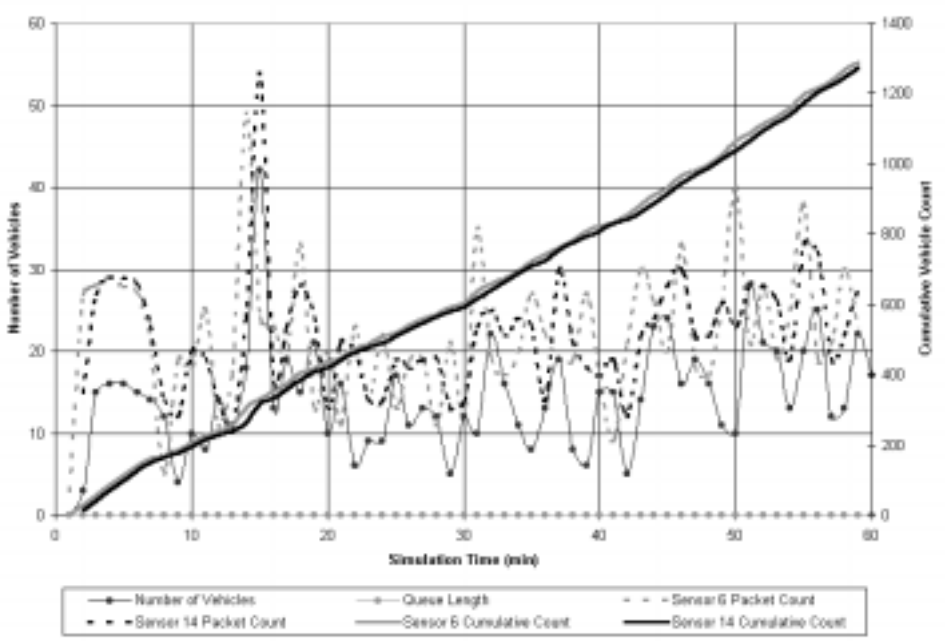

Figure 6: Segment 213 Chart

within the middle of the CA-T network. Because segment 212 and 213 have multiple lanes, they have large input and output capacities. Segment 212 accepts input from several different sources which accounts for the wavy character of its density and flows. Since the output from segment 209 is negligible compared to that of segment 212, segment 213 shows the same waviness.

3.4 Sensitivity Testing

We tested the sensitivity of the simulator to different parameters by setting up a series of test cases. Each test case varies the update and advance interval of the simulation while keeping all other parameters constant (see Table 2). In each case, the simulation was run on the CA-T network with a demand of 18,244 packets for a simulation time of one hour.

We calculated the correlation coefficient for each possible pairing of test

Case Number	Update Interval (sec)	Advance Interval (sec)
1	300	150
2	120	60
3	120	30
4	60	60
5	60	30
6	30	15
7	15	15

Table 2: Sensitivity Test Case Matrix

cases. Since the queue length and density are written as the instantaneous values at the end of each update interval, the amount of data varies between test cases. We decided to sample the data from the file with the smaller update interval at points equal to the update intervals in the file with the larger update interval. For example, if we were comparing a file with an update interval of 30 seconds and a file with an update interval of 60 seconds, we would only use every other update interval in the former file to compute the correlation coefficient. This method allows us to compare all of the test cases, except for the cases where the update intervals are 300 seconds and 120 seconds. In these cases, we cannot compare the files because their update intervals are not evenly divisible. The flow, speed, and travel time information files are not dependent on the update intervals so there is no need to sample data from them.

	120-30	60-60	60-30	30-15	15-15
300-150		0.468	0.444	0.393	0.373
120-60	0.697	0.545	0.565	0.458	0.419
120-30		0.645	0.680	0.464	0.432
60-60			0.755	0.684	0.634
60-30				0.690	0.623
30-15					0.941

Table 3: Density File Correlation Coefficients

For the queue length and density files (see Tables 3 and 4), the correlation coefficients show that the test cases with the most similar update intervals

	120-30	60-60	60-30	30-15	15-15
300-150		0.224	0.196	0.186	0.180
120-60	0.551	0.314	0.248	0.182	0.170
120-30		0.389	0.475	0.270	0.255
60-60			0.535	0.361	0.336
60-30				0.686	0.622
30-15					0.940

Table 4: Queue Length File Correlation Coefficients

are the most strongly correlated. This is to be expected because the data reported is a snapshot of the simulation at the end of the update phase. There does not appear to be a pattern with respect to the advance intervals. For example, the density table shows that the correlation between the 120-60 file and the 15-15 file is weaker than that between the 120-30 file and the 15-15 file. Yet the correlation between the 60-60 file and the 15-15 file is stronger than that between the 60-30 file and the 15-15 file. In the queue table, the 120-30 file has a stronger correlation to the 60-60 file than the 120-60 files does, despite the fact that the 120-60 and 60-60 files are more closely related with respect to the advance interval.

In general, the correlation coefficients for the density and queue length files show a moderate correlation. The 300-150 test case shows the weakest correlation in almost every case, owing to the large differences in granularities. Although the density and queue length files will not be used as input to the OD estimation module or to the guidance generation module, the correlation between these files can be used in conjunction with the timing data to choose the best simulation with regards to accuracy and speed of simulation.

The flow information files (see Table 5) show a very strong correlation. The correlation coefficient for these files was calculated by comparing the time recorded when packets cross sensors with respect to each unique packet and sensor pair. There appears to be no pattern with respect to update intervals, which is reasonable because the files are not dependent on update intervals. The files are weakly dependent on advance intervals, because advance intervals affect the granularity of the actual simulation algorithm. For example, the correlation coefficient for the 60-60 and 60-30 files is 0.879, while the correlation coefficient for the 60-30 and 120-30 is 0.893. The entire row

	120-60	120-30	60-60	60-30	30-15	15-15
300-150	0.821	0.854	0.820	0.799	0.780	0.774
120-60		0.880	0.839	0.806	0.796	0.793
120-30			0.858	0.893	0.862	0.861
60-60				0.879	0.894	0.853
60-30					0.900	0.890
30-15						0.978

Table 5: Flow Information File Correlation Coefficients

associated with the 120-30 shows the same trend; the files with the most similar advance intervals are better correlated, with little regard to update interval.

Since DynaMIT is a macroscopic simulator, we are interested mainly in aggregate measures of travel times. The travel times of individual packets are inconsequential in comparison with the travel times of aggregate groups of packets traveling on the same link. Thus, we calculated the correlation coefficients for travel times based on the average travel times of all packets on a link.

	120-60	120-30	60-60	60-30	30-15	15-15
300-150	0.824	0.721	0.789	0.653	0.593	0.597
120-60		0.853	0.847	0.728	0.589	0.581
120-30			0.790	0.840	0.657	0.640
60-60				0.852	0.822	0.806
60-30					0.871	0.851
30-15						0.988

Table 6: Travel Time Information File Aggregate Correlation Coefficients

The correlation in the travel time files (see Table 6) is better than the correlation of the queue length and density files, which is important because the travel times will be used by the guidance generation module. Similarly to the queue length and density files, the travel time correlation coefficients appear to be affected more by the update interval than by the advance interval.

	120-60	120-30	60-60	60-30	30-15	15-15
300-150	0.385	0.284	0.347	0.267	0.232	0.238
120-60		0.470	0.404	0.357	0.253	0.252
120-30			0.383	0.467	0.294	0.289
60-60				0.441	0.401	0.391
60-30					0.547	0.518
30-15						0.693

Table 7: Travel Time Information File Disaggregate Correlation Coefficients

As an interesting side note, we calculated the correlation coefficients for travel times at a much more disaggregate level by comparing the travel time for each link and packet pair. Although the correlation coefficients degrade substantially (see Table 7), it is worthwhile to note that pairs with a close correspondence in advance and update intervals (such as the 60-30/30-15 and 30-15/15-15 files) still have reasonably strong correlation coefficients.

3.5 Validity Checking Results and Discussion

The series of validity checks yielded five anomalies in the simulator. Three were errors that have been corrected; the remaining two were the result of the traffic model used in the simulator. They have also been corrected to yield better results.

The first anomaly was that nine segments had densities significantly higher than 200 vehicles per meter. We created density graphs for the segments (see Figure 7 for an example) with the highest densities and for the upstream and downstream segments surrounding these segments. The graphs showed that the density increased sharply in the presence of a queue and then decreased sharply when the queue dispersed. This led us to believe that the queue length was not being considered when the simulator allowed packets to enter the segment. The error was found in the `canGoTo` method, which is used during the advance phase to determine if packets can enter the next segment. The method was not using the length of the queue in its calculation of how many packets were on the segment. The addition of the queue length into the calculation fixed this error.

We found the next error when we tried to import flow information into the

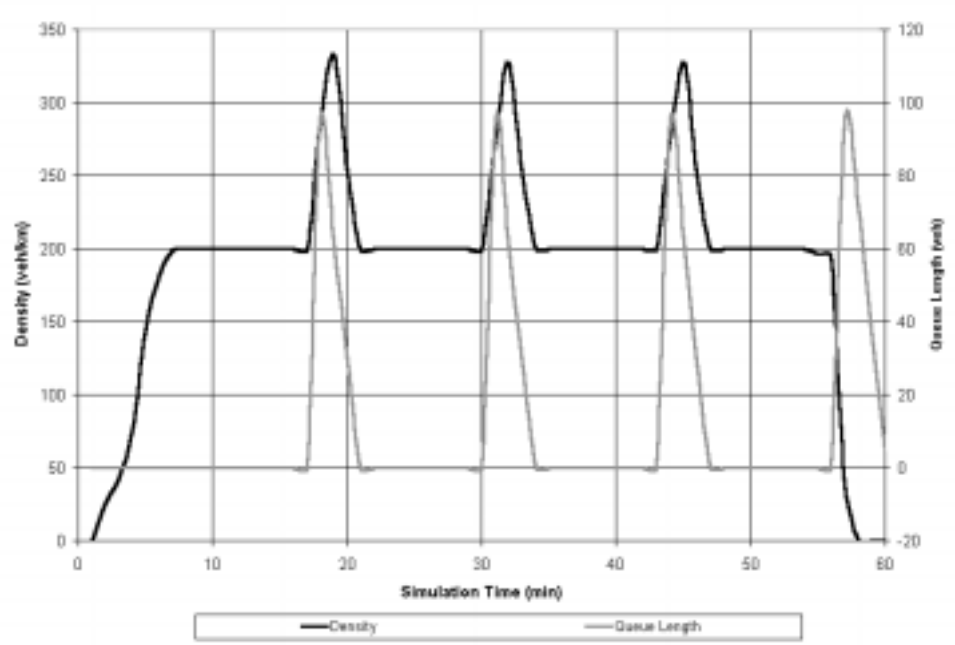

Figure 7: Segment 209 Chart, Incorrect Density

database. The flow information file prints a sensor identification number, a packet identification number, and the simulation time when the packet crossed the sensor. The information is stored in a table in the database, in which a sensor number and packet number combination must be unique. When we tried to import the flow information into the database, a primary key violation occurred because each sensor number and packet number combination was not unique.

This error was also evident in graphs that showed the flow out of the segment was greater than the flow into the segment. The anomaly was caused by the `moveToNextSegment` method, which is used during the advance phase to move a packet to the next segment in its path. We discovered that the `moveToNextSegment` method does not advance the packet to the next segment if the next segment is full. In this case, the packet is replaced on the previous segment in the same position before `moveToNextSegment` was called. The error occurred when the simulator tried to calculate the final position of the packet to determine if it had crossed any sensors. Instead

of calculating the final position as the packet's previous position, the final position was set to zero. The packet remained in the same position, but the sensor, which was placed near the very end of the segment (see Figure 3), was triggered and recorded the time when the packet "crossed" it. Each time this sequence was repeated, the sensor recorded the packet and the simulation time. A simple change to the final position calculation rectified this error.

The third anomaly was discovered immediately after the second had been corrected (see Figure 8). The difference between the cumulative flow into the segment and the cumulative flow out of the segment should be equal to the density of the segment. For example, at simulation time 40 minutes, a total of 665 vehicles have crossed the upstream sensor of segment 282. At the same time, a total of 429 vehicles have crossed the downstream sensor and have moved onto the next segment. The difference between these two numbers should be equal to 65, the density of the segment at time 40 minutes. We found that the difference between the flow into and out of segment 282 was 171 packets greater than the density of the segment. By using the database, we found that these packets crossed the upstream sensor but never crossed the downstream sensor. We also found that these same packets crossed the upstream sensor of the next segment on their path.

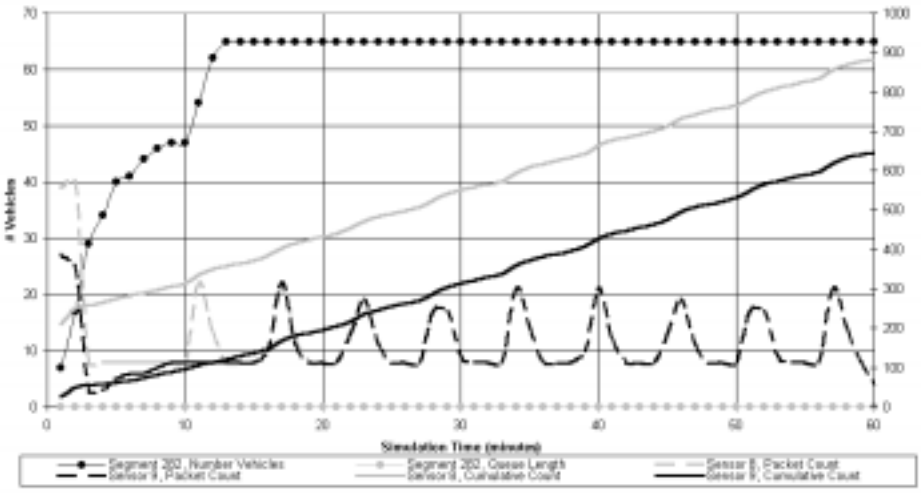

Figure 8: Segment 282 Chart, Incorrect Flows

Eventually we determined that the error was caused when the packets were placed on a virtual queue. The final position of the packets on the previous segment was never set; they never crossed the downstream sensor. This error was corrected by setting the final position of the packet on the previous sensor to be zero when the packet is placed on the virtual queue.

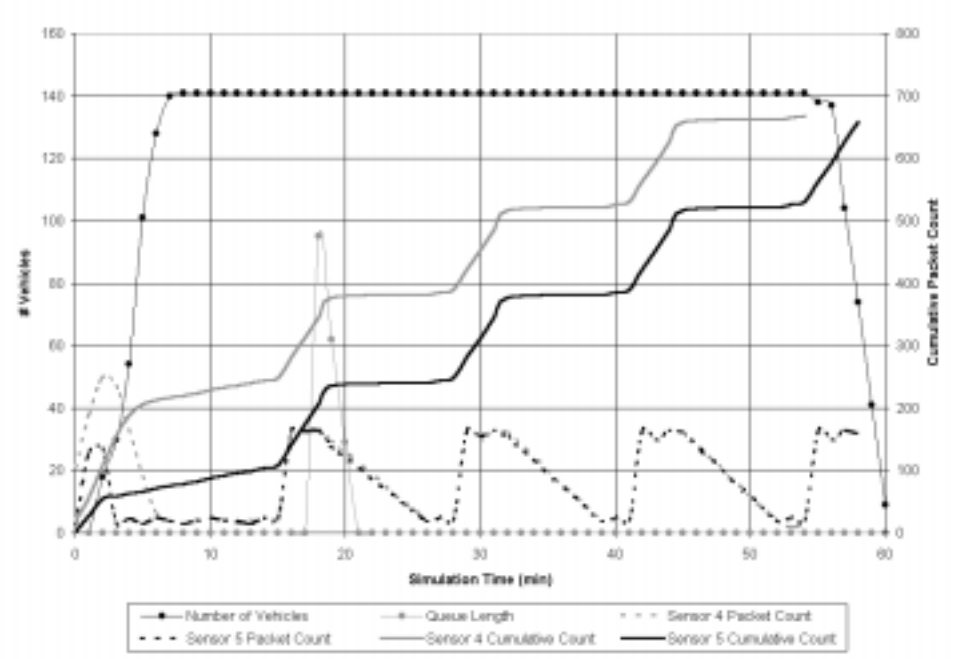

Figure 9: Segment 209 Chart, Uneven Flow

The next anomaly is exhibited most clearly by segment 209 (see Figure 9). Instead of having an even flow in and out, the flow shows a wave-like behavior. At the high points, the flow is at capacity for the segment (33 packets per minute) and at the low points, it is nearly zero. Except for a queue that lasts for a duration of about three minutes, the segment has no queues and there appear to be no problems with the output capacity of the segment. Segment 209's downstream segment has a large input capacity, which its density does not even approach. Therefore, the flow out of the segment should be nearer to capacity.

The unique geometry of this segment causes the anomaly (see Figure 3).

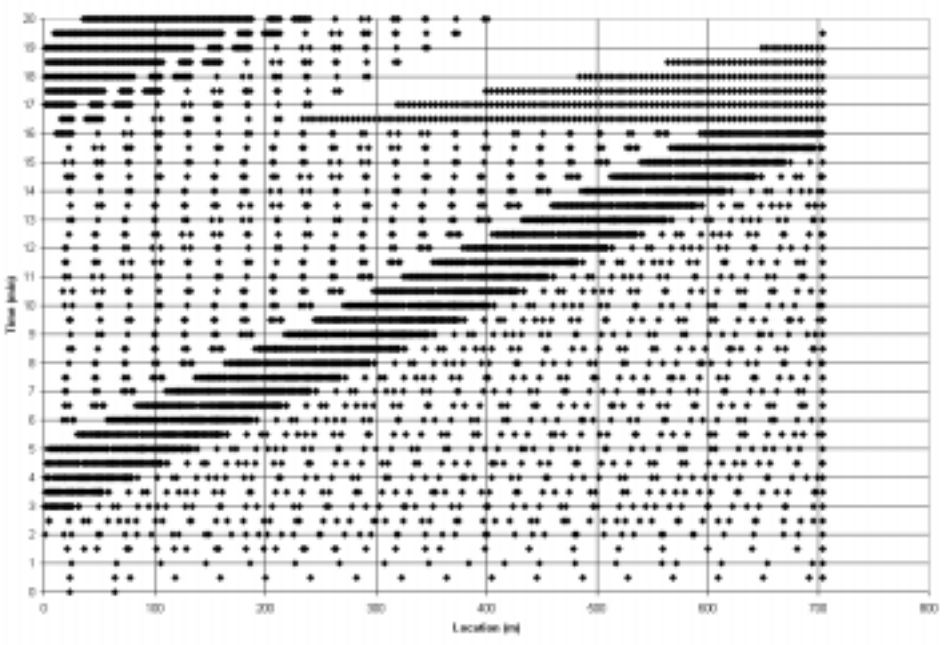

Figure 10: Location of Packets on Segment 209, Old Version

Segments 206 and 207 merge together to form segment 208, which has two lanes. The two lanes in segment 208 merge together into one lane in segment 209. In the simulator model, packets are represented as points, rather than one-dimensional lines with a length. Because of this, the packets from the two lanes in segment 208 bunch up very closely in segment 209, filling the capacity of the segment. However, the density is not evenly distributed. Most of the packets are at the upstream end, while only a few are at the downstream end (see Figure 10). The velocity of the packets is determined by the density, so the packets move very slowly. It takes the packets approximately thirteen minutes to travel the length of segment 209 at the minimum speed allowed by the simulator. During the time these packets are on the segment, only a few packets are allowed to enter the segment because only a few are leaving and the segment is nearly full to capacity. At the end of thirteen minutes, the large group of packets reaches the end of the segment and exits. Suddenly the density of the segment drops, and the segment allows another large group of packets to enter the segment, repeating the cycle.

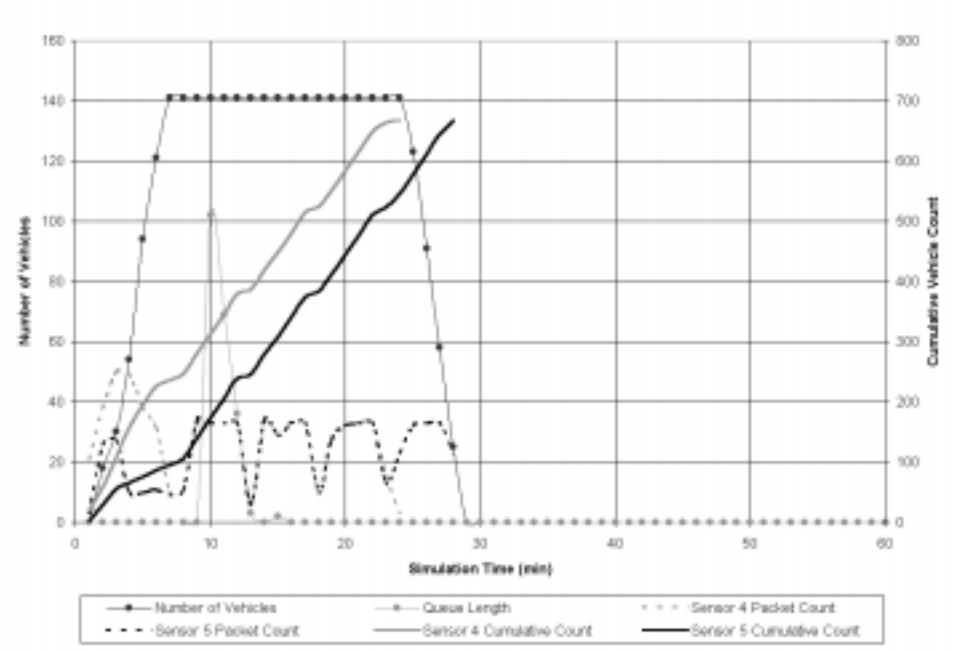

Figure 11: Segment 209 Chart, Minimum Speed 2.25 m/s

We tested our theory by tracing packets through the segment. By looking at the packets and their locations, we realized the packets were traveling in large groups through segment 209. We devised another test for the theory by changing the minimum speed allowed by the simulator. We increased the minimum speed from 0.8940 meters per second to 2.25 meters per second. This decreased the time needed for the packets to travel across segment 209 to five minutes. As we expected, the "wavelength" of the flow graphs was shortened to roughly five minutes (see Figure 11).

We corrected this problem by changing the simulator model to reflect the physical length of the packets. This was accomplished by adding two attributes to the `dtaSegment` class. The new attributes monitor the time when the last packet was loaded onto the segment and the rate at which new packets can be loaded onto the segment. The `loadNewPackets` and `canGoTo` methods were modified to use these two new attributes when determining if a packet can be loaded onto or can advance to the next segment. This modification fixed the wavelike behavior of the flows (see Figure 5) and

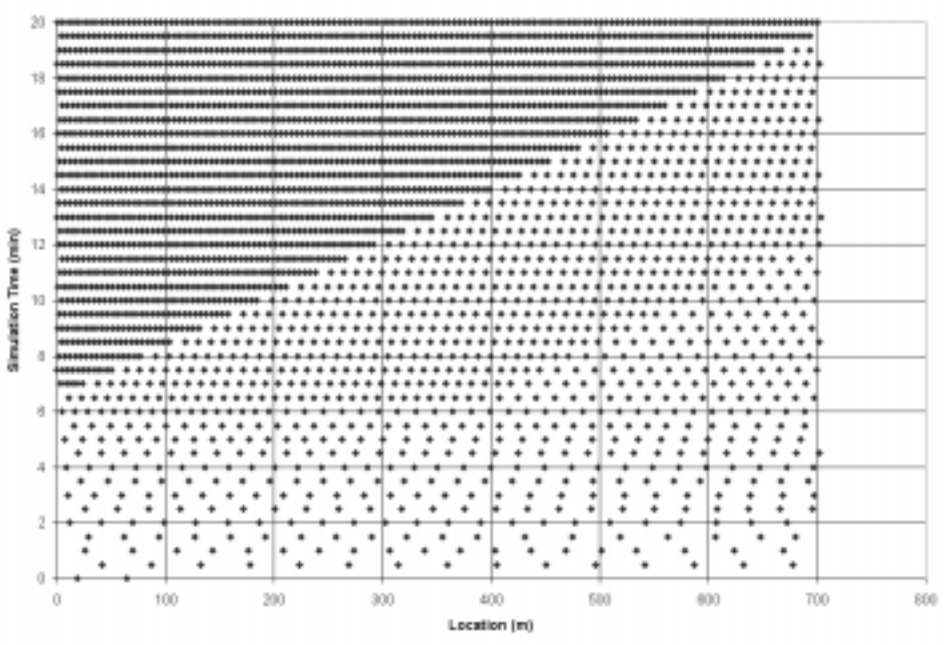

Figure 12: Location of Packets on Segment 209, Corrected Version

the distribution of packets on the segment (see Figure 12).

The final anomaly was discovered during a set of runs of various grid networks. We noticed grids with a demand of 3000 packets per hour per loader were simulated in less time than the same grid with a demand of 2000 packets per hour per loader. Closer inspection revealed that grids with a demand of 3000 packets were loading significantly fewer packets than the grids with smaller demand. We tested the simulator with a 2x2 grid, which showed that the links were receiving unequal numbers of packets; three received a similar number of packets, but the fourth received significantly fewer packets. The fourth link and the first link were connected to the same loader, which led us to believe that the simulator was not programmed to consider the case where a loader connects to two different links.

The `loadNewPackets` method iterates through a list of packets that are waiting at a loader to enter the network. The method uses several criteria to decide when to load a packet. The speed of packets on the segment where

the packet will be loaded, the queue length in the lanes of the segment, the density of the segment, and the time when the last packet was released are all considered. When calculating these values, the `loadNewPackets` algorithm assumed that each loader was connected to only one link. When deciding whether to load more packets the loader considered only the first link, which was full, and was unable to realize that the second link was empty. We were able to correct this behavior with several minor changes to the algorithm and the addition of an attribute to the `dtaSegment` class. The new attribute keeps a record of the time when the last packet was released onto the segment. Subsequent runs of the grid networks verified that the code had been corrected.

4 Performance Testing

Since the time performance of a prediction model like DynaMIT is critical, the speed of the simulation and, to a lesser extent, the memory requirements of the simulation become extremely important. Our performance tests accomplish several objectives. First, the tests ensure that the simulator is running within an acceptable amount of time. Second, the results of the tests allow the user to choose a granularity of simulation, knowing the tradeoffs between granularity and speed. Finally, the tests can uncover potential problems in the simulator and areas where the simulation speed may be improved.

We ran four different categories of tests. The first and second categories tested the performance of the simulator on the grid network and the CA-T network, respectively. The final two categories used two commercial software packages to evaluate the performance of the simulator. We used Purify to identify possible memory leaks and Quantify to profile the computational characteristics of the simulator.

4.1 Grid Network

Testing the grid network allowed us the most flexibility in varying the topology and packet demand components of the simulator. We ran three types of tests: comparing the effects of different grid sizes, different grid sizes and packet demand rates, and different grid sizes and granularities. The tests

were run on a Sun UltraSPARC workstation, with all output to disk suppressed. In each case, the output capacities of the lane groups were set so that no traffic queues formed during the simulation.

4.1.1 Test Matrix

Test 1 evaluates the effects of varying the grid size. Square grids with $m = n = 5, 10, 15, 20, 25, 30,$ and 50 columns and rows were tested.

Test 2 evaluates the effects of packet demand and variations in grid size. Square grids with $m = n = 10, 20, 30,$ and 50 columns and rows were tested for demands of $100, 500, 1000, 2000,$ and 3000 vehicles per hour.

Test 3 evaluates the effects of granularity and variations in grid size. Square grids with $m = n = 10, 20, 30,$ and 50 columns and rows were tested with update intervals of 120 seconds and 60 seconds. For update intervals of 120 seconds, the grids were tested with $1, 2,$ and 4 advance intervals per update interval. For update intervals of 60 seconds, the grids were tested with 1 and 2 advance intervals per update interval.

Parameter	Test 1	Test 2	Test 3
Link Length (m)	500	500	500
Loader Input/Output Flow (veh/sec)	8	8	8
Free-flow Speed (m/sec)	10	10	10
Jam Density (veh/m)	0.1243	0.1243	0.1243
Alpha	2.8	2.8	2.8
Beta	5.0	5.0	5.0
Segment Input Flow (veh/hr)	1200	7500	1200
Lane Group Output Flow (veh/hr)	1200	7500	1200
Packet Demand Per Loader (veh/hr)	800	varies	800
Packet Length (m)	5.0	5.0	5.0
Simulation Time (sec)	3600	3600	3600
Update Interval (sec)	120	120	varies
Advance Interval (sec)	60	60	varies

Table 8: Grid Network Performance Test Matrix

4.1.2 Results and Discussion

Grid Size	Number of Vehicles on Grid	Average Run Time (sec)
5 x 5	444	4.01
10 x 10	2000	12.31
15 x 15	4667	24.39
20 x 20	8444	40.07
25 x 25	13,333	58.01
30 x 30	19,333	79.31
50 x 50	54,444	180.08

Table 9: Varying Grid Size

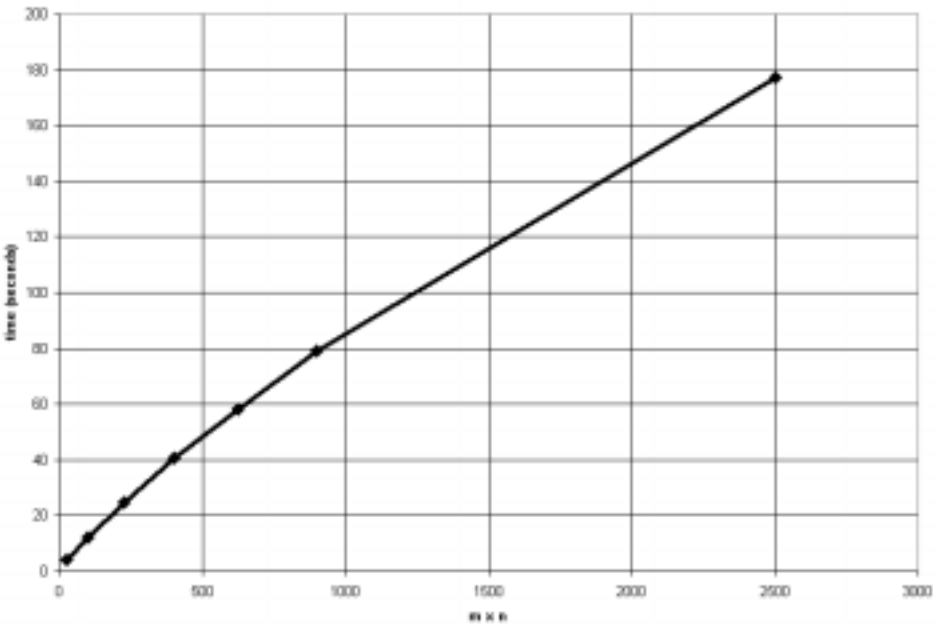

Figure 13: Varying Grid Size

The results of the first performance test (see Table 9 and Figure 13) show that the run time of the simulator appears to vary linearly with the total

size of the network. With times ranging from four seconds to three minutes for a simulation of one hour, the run times are reasonable for uncomplicated networks that do not develop traffic queues.

Grid Size	Packet Demand Per Loader (veh/hr)	Maximum Number of Vehicles on Grid	Average Run Time (sec)
10 x 10	100	278	1.59 (1.51)
10 x 10	500	1389	7.03 (6.76)
10 x 10	1000	2778	16.12 (11.24)
10 x 10	2000	5556	43.55 (28.32)
10 x 10	3000	8333	392.83 (37.04)
20 x 20	100	1111	5.74 (10.19)
20 x 20	500	5556	22.64 (47.10)
20 x 20	1000	11,111	50.61 (64.15)
20 x 20	2000	22,222	131.34 (121.11)
20 x 20	3000	33,333	751.84 (155.97)
30 x 30	100	2500	12.26 (44.75)
30 x 30	500	12,500	45.55 (197.95)
30 x 30	1000	25,000	98.81 (253.30)
30 x 30	2000	50,000	252.38 (412.77)
30 x 30	3000	75,000	1136.98 (528.28)
50 x 50	100	6944	32.78 (322.10)
50 x 50	500	34,722	103.88 (1369.27)
50 x 50	1000	69,444	214.57 (1997.78)
50 x 50	2000	138,889	526.21 (2515.60)
50 x 50	3000	208,333	not enough space to store input files (3077.42)

Table 10: Varying Grid Size and Packet Demand

By looking at the variation of grid size while the number of vehicles loaded remains constant, the results from the second test (see Table 10 and Figure 14) show that the run time roughly doubles each time the grid size increases. Within each grid size, the run time appears to vary almost linearly with the number of vehicles loaded per hour per loader, with one exception. A large jump in the run time occurs when the number of vehicles

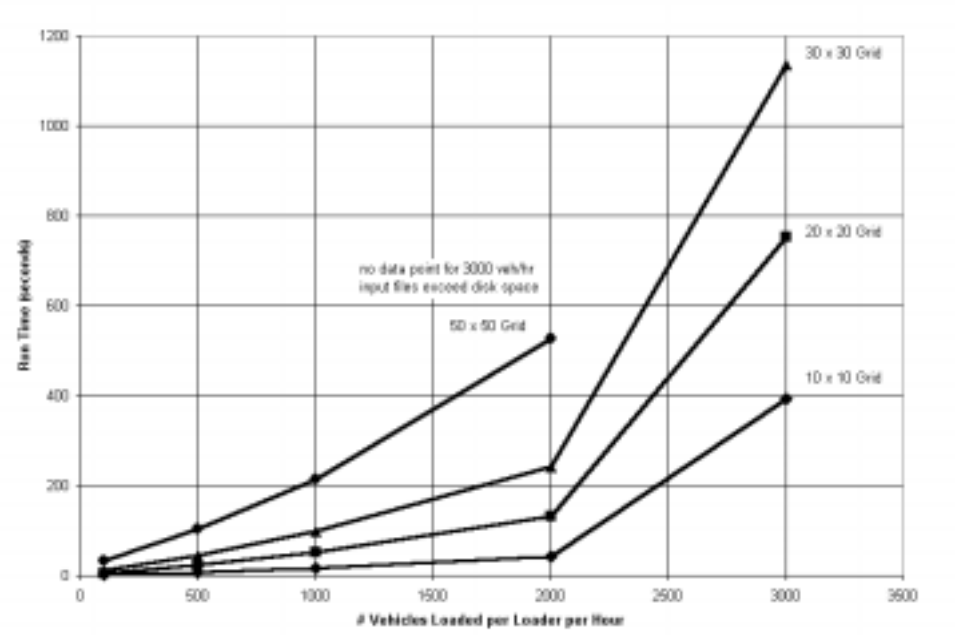

Figure 14: Varying Grid Size and Packet Demand

loaded reaches 3000. With the use of Purify (see Section 4.3), we discovered that this increase was due to a memory leak in the part of the simulator algorithm that advances packets to the next segment. The leak caused the program to lose so much memory that the workstation was forced to begin swapping memory, which slowed the run time considerably.

The numbers in parentheses show the run times recorded by the corrected version of the simulator. In some cases, the corrected version of the simulator runs slower than the older version (note the times for the 20x20 and larger grids). This is due to changes in the way the path topology is processed, which causes the difference in times to increase as the grid size increases. These changes were made in the simulator code between the tests. However, the corrected version does not display the same non-linear increase in run times at a demand of 3000 packets per hour. The run times for the corrected version are linear throughout the test grid.

Finally, the results of the third test, varying grid size and granularity (see

Grid Size	Number of Advance Intervals per Update Interval	Update Interval of 120 sec, Average Run Time (sec)	Update Interval of 60 sec, Average Run Time (sec)
10 x 10	1	9.22	12.05
10 x 10	2	12.43	17.32
10 x 10	4	17.16	26.91
20 x 20	1	28.23	38.2
20 x 20	2	39.16	56.71
20 x 20	4	56.84	92.02
30 x 30	1	54.71	74.44
30 x 30	2	77.94	113.69
30 x 30	4	113.67	187.56
50 x 50	1	139.33	169.28
50 x 50	2	174.20	252.98
50 x 50	4	266.82	virtual memory exceeded

Table 11: Varying Grid Size and Granularity

Table 11, Figure 15, and Figure 16), showed that the run time of the simulator varies linearly as the number of advance intervals per update interval changes. In the last test case, a 50 x 50 grid with four advance intervals per update interval, the virtual memory of the workstation was exceeded and the simulator was unable to finish. This also appeared to be the result of a memory leak in the simulator code, which was exacerbated by the fine granularity of the simulation. With the use of Purify (see Section 4.3), we determined that the same memory leak found in the second test was responsible for the results in this test. Since the packet advancing method is called for each advance interval, increasing the number of advance intervals increases the number of calls to the method.

4.2 CA-T Network

The CA-T network comparisons test the supply simulator on a network modeled after a real traffic scenario. Although this limits the flexibility in varying the topology and packet demand components of the simulator, it is an important component of the test matrix because it tests the simulator under more complex conditions than can be generated for a grid network.

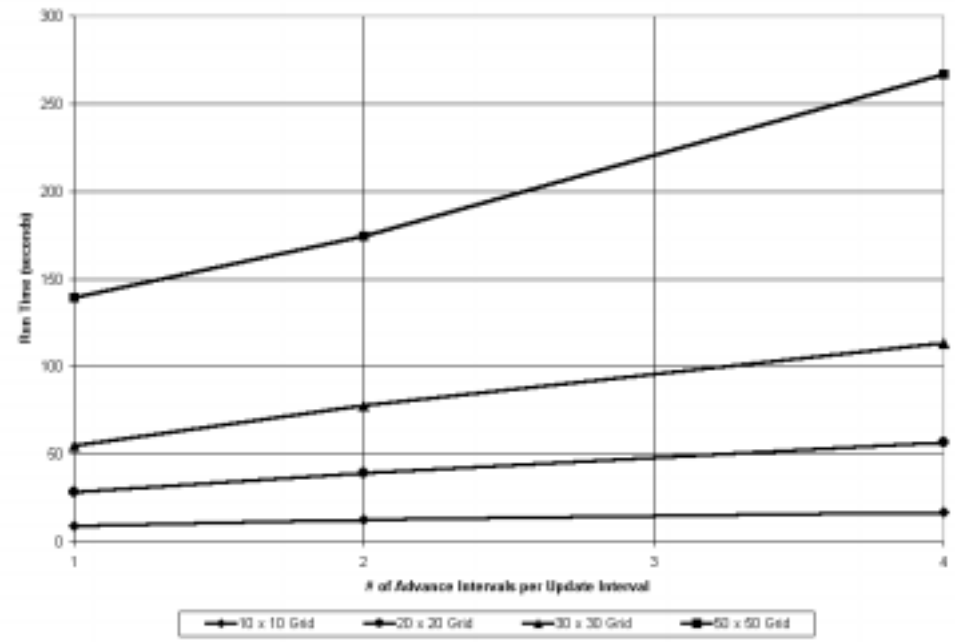

Figure 15: Varying Granularity in Grid Network, Update Interval 120 seconds

We were able to vary packet demand to a small extent, and we were able to test granularity thoroughly. The tests were run on a Sun UltraSPARC workstation, with all output suppressed.

The standard packet demand file contains 18,244 packets. The file was edited by hand to produce two more files, one with 10,000 packets and one with 36,488 packets. Each of the different packet files were tested with granularities of two, four, and eight advance intervals per update interval of 120 seconds and granularities of two and four advance intervals per update interval of 60 seconds.

The results (see Table 12 and Figure 17) show that run time varies linearly with granularity, except for the case in which packet demand was 36,488. In this case, the increased number of packets on the network causes a large amount of queuing which slows the simulation as the granularity becomes finer. Once again, memory leaks in the code became apparent when the

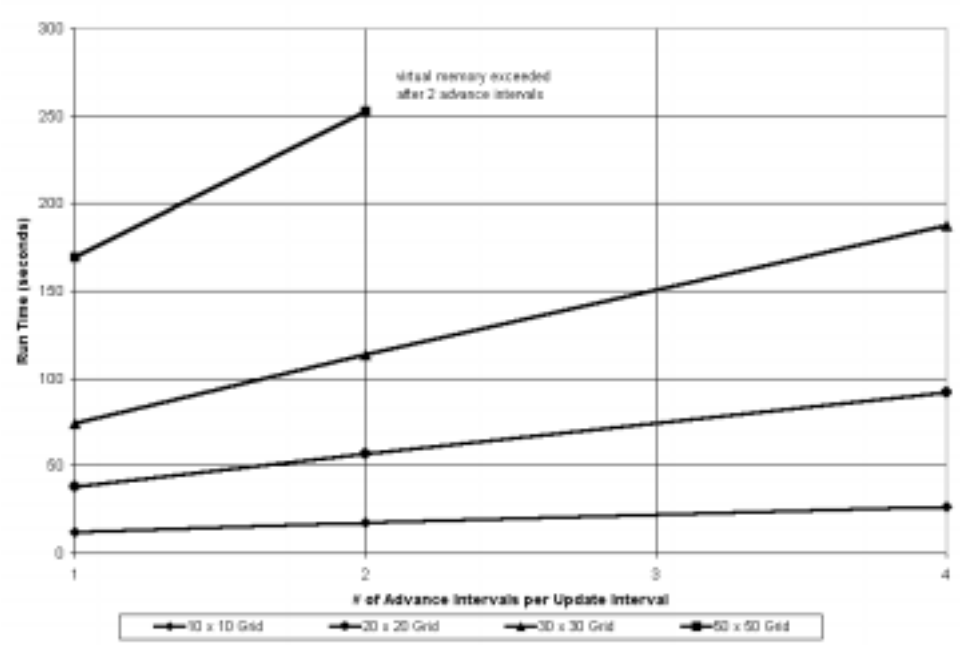

Figure 16: Varying Granularity in Grid Network, Update Interval 60 seconds

machine exceeded its virtual memory during two tests: with four advance intervals per update interval of 60 seconds and with eight advance intervals per update interval of 120 seconds. The numbers in parentheses in the table show the run times of the simulator once these memory leaks were fixed.

An interesting trend in the run times is revealed by comparing simulations with the same number of total iterations. For example, a simulation with an update interval of 120 seconds has 30 update phase iterations within an hour simulation. If there are eight advance intervals per update interval, the simulation contains a total of 240 advance phase iterations. Thus, a simulation with an update interval of 120 seconds, with eight advance intervals per update interval, has the same number of total advance phase iterations as a simulation with an update interval of 60 seconds, with four advance intervals per update interval. The run times for simulations with the same number of total advance iterations are remarkably similar for each of the cases presented in Table 12.

Packet Demand	Number of Advance Intervals per Update Interval	Update Interval of 120 sec, Average Run Time (sec)	Update Interval of 60 sec, Average Run Time (sec)
10,000	2	71	98
10,000	4	91	136
10,000	8	127	
18,244	2	125	167
18,244	4	159	253
18,244	8	232	
36,488	2	193	248
36,488	4	237	virtual memory exceeded (394)
36,488	8	virtual memory exceeded (410)	

Table 12: Varying Granularity in CA-T Network

4.3 Purify

Purify was used to track memory leaks in the simulator code. Purify found roughly seven megabytes worth of potential memory leaks in the original code; several minor changes made to the code reduced the amount to three megabytes of potential leaks. The leaks were found in two methods that control the movement of packets during the advance phase of the simulator algorithm.

The leaks became apparent when three different runs of the simulator exceeded the Sun UltraSPARC workstation's virtual memory, causing the simulation to end prematurely. The first run was on a 50 x 50 grid network with four advance intervals per update interval of 60 seconds. The other two runs were on the CA-T network with four advance intervals per update interval of 60 seconds and eight advance intervals per update interval of 120 seconds, both with a demand of 36,488 packets. Additionally, the simulator displayed non-linear behavior in run times when the number of packets loaded per hour per loader exceeded 2000 for several different grid sizes. After the leaks were removed, the runs were simulated again. The simulator finished the runs in all cases (see numbers in parentheses in Table 10 and Table 12).

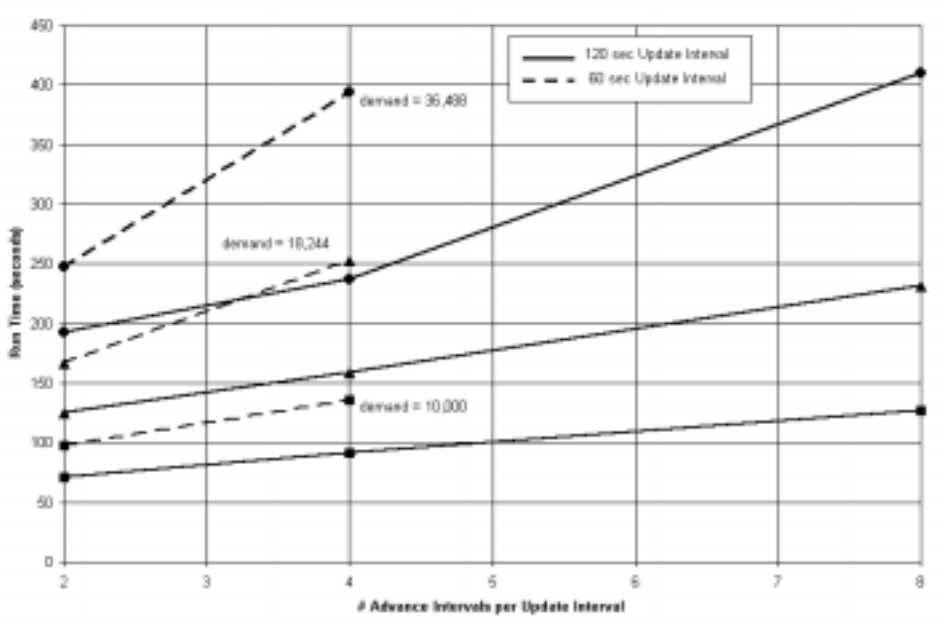

Figure 17: Varying Granularity in CA-T Network

4.4 Quantify

The final step of the performance testing process was to develop a profile of the CPU usage for the simulation algorithm. We used Quantify to generate this profile. First, the simulator was tested with file output on and with file output suppressed on the CA-T network (see Table 13 and Figure 18). The simulation was run for one hour on a Hewlett-Packard 9000 with an update interval of 60 seconds, an advance interval of 30 seconds, and a demand of 18,244 packets. The simulator was also tested on a 10x10 and a 30x30 grid network with packet demands of 100 and 1000 vehicles per hour per loader on the same machine. The parameters for the second timing test (see Table 8) were used for the simulation.

The Quantify profile for the simulation with output (see Table 13 and Figure 18) shows that writing the output to disk uses a significant amount of CPU time. The output of this particular simulation run occupies approximately 4.5 megabytes of disk space, which takes 54.5 seconds (more than

Task (run time, sec)	With Output	Output Suppressed
Run Time (sec)	171.36	112.08
Path Following for Each Packet	35.63% (61.1)	54.47% (61.0)
Write Output Files	31.82% (54.5)	-
Read Input Files	3.20% (5.5)	5.85% (6.6)
Sort Packets At Nodes	8.34% (14.3)	12.75% (14.3)
Load Packets Onto Network	1.34% (2.3)	2.05% (2.3)
Advance Packets Through Network	1.71% (2.9)	2.62% (2.9)
Other	17.96% (30.8)	22.26% (24.9)

Table 13: Quantify Profile, CA-T Network

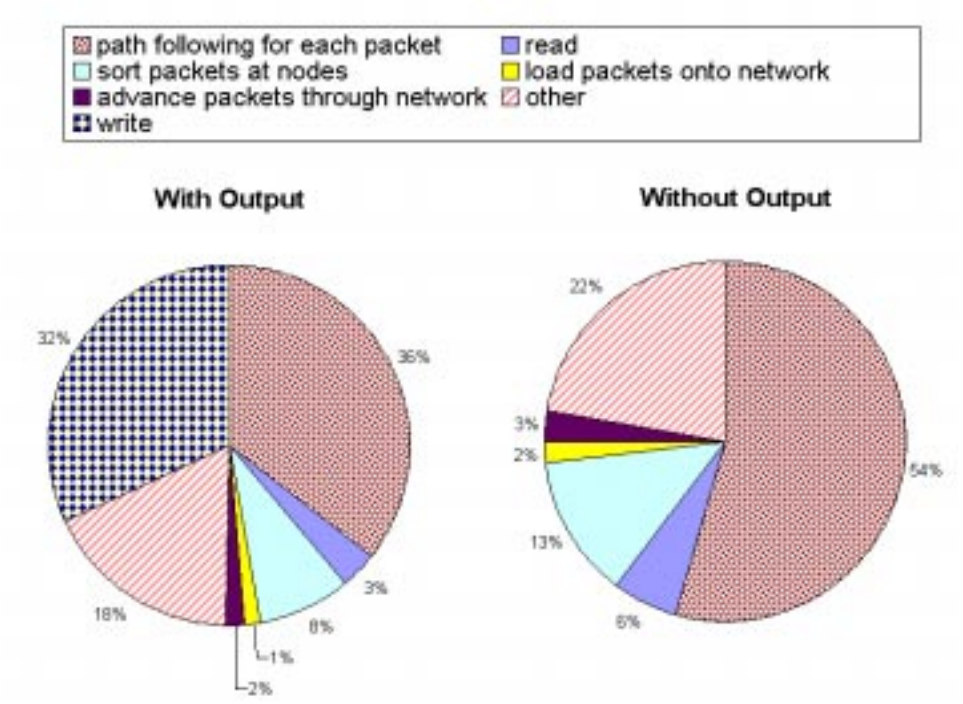

Figure 18: Quantify Profile, CA-T Network

30%) of CPU time to write. The queue length, density, and speed information files are not used by any other modules; they constitute about one

megabyte of disk space. The travel time and flow information files, which will be used by the guidance generation and OD estimation modules, use the remaining 3.5 megabytes of space. Therefore, the time to output information for the other two modules is roughly 42 seconds.

The biggest percentage of CPU time was used to generate the next path and link for each packet. However, this is a temporary method to replace the network topology component for debugging purposes; it will not be used in the final version of the simulator. Of the methods that will remain in the final version, sorting packets at each node uses a significant amount of CPU time. Streamlining this method may result in a small gain in processing time. The remaining methods in the simulator all use less than 5% of the total CPU time; optimizing these methods will probably not yield significant timing gains.

Generating and writing the output files to disk seems the likeliest candidate for improvements to the simulation speed. Using only the output needed by other modules will reduce CPU time. Since output is written every update interval, increasing the update interval will also speed up the simulation. It may also be possible to optimize the reporting algorithm itself.

Task (run time, sec)	10x10 Grid 1000 veh/hr	30x30 Grid 1000 veh/hr	30x30 Grid 100 veh/hr
Run Time (sec)	27.820	536.359	97.099
Path Following For Each Packet	22.77% (6.33)	75.16% (403.1)	70.66% (68.7)
Read Input Files	25.37% (7.06)	1.56% (8.37)	7.18% (6.98)
Load Packets Onto Network	6.73% (1.87)	0.99% (5.31)	1.17% (0.16)
Sort Packets At Nodes	3.81% (1.06)	1.11% (5.95)	0.62% (0.60)
Advance Packets Through Network	3.58% (1.00)	1.25% (6.70)	1.18% (1.15)
Other	37.74% (10.5)	19.93% (106.9)	20.19% (19.6)

Table 14: Quantify Profile, Grid Network

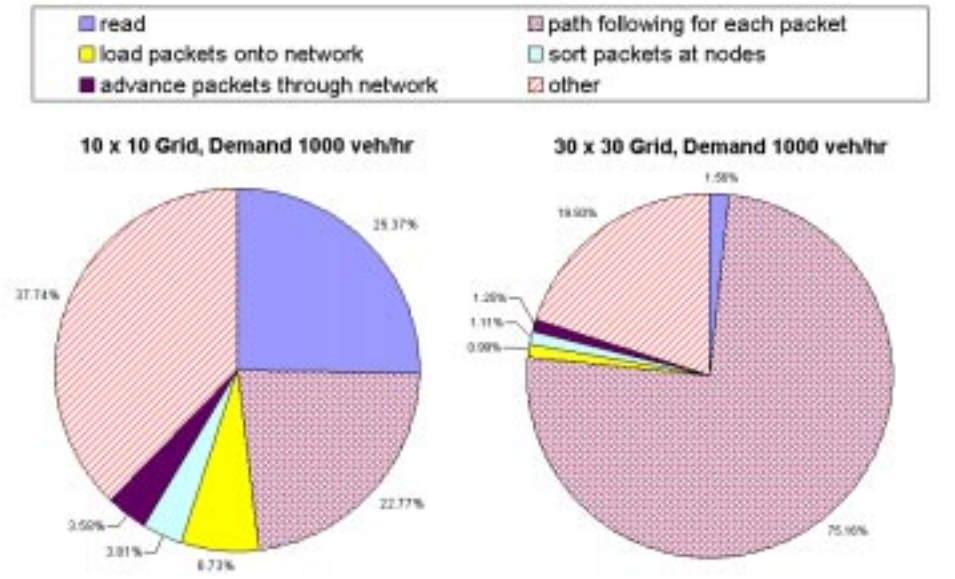

Figure 19: Quantify Profile, Grid Size

The Quantify profile for the grid networks show that generating the next path and link for each packet uses a significant amount of CPU time for all of the cases. As mentioned above, this method is a temporary debugging method that will not be included in the final simulator.

The CPU time used to read input files was roughly constant among the three networks and very similar to the read times for the CA-T network simulations. The remaining methods each use less than 5% of the total CPU time, with the exception of loading packets for the 10x10 case.

The actual run times for the method to sort packets at nodes show that some improvement in this method may be useful for very large networks with high demands. To illustrate, the 10x10 grid case will load a total of 20,000 packets onto the network, the 30x30 grid with a demand of 1000 vehicles per hour loads a total of 60,000 packets, and the 30x30 grid with a demand of 100 vehicles per hour loads a total of 6000 packets. The increase from 6000 total packets to 20000 total packets less than doubles the time used to sort the packets. However, the increase from 20000 total packets to 60000 total packets results in a run time that is six times slower.

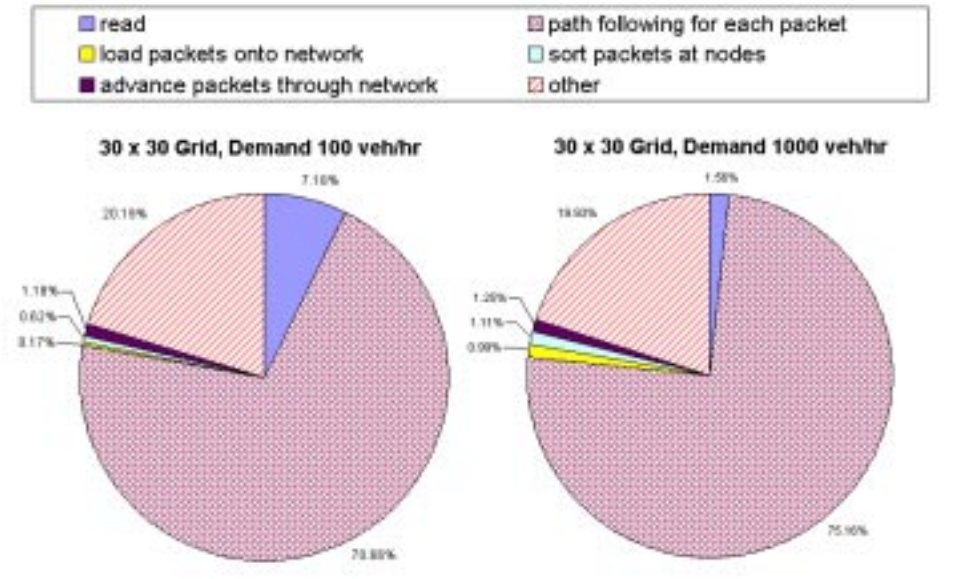

Figure 20: Quantify Profile, Packet Demand

5 Conclusions and Future Work

The supply module has been tested for a variety of network topologies, packet demand scenarios, and granularities. The slowest one-hour simulations of the CA-T network ran in just under seven minutes. The slowest simulation of the grid network ran for roughly fifty minutes. This one-hour simulation for a 50x50 grid with a packet demand of 3000 vehicles per hour per loader tested the limits of the supply module. However, all of the simulations involving 30x30 and smaller grids ran in under ten minutes. The performance tests show that the simulator is a viable real-time system for these grids and for the CA-T network. Optimization of the output functions and sorting methods will further enhance run time performance.

Special attention should be paid when the path topology component interface is connected to the supply module. Streamlining the interface methods between the two components could result in huge gains in processing time;

in all test cases, the Quantify profile showed that the temporary methods used to mimic this interface consumed the largest percentage of CPU time. The 50x50 grid network may even become a viable real-time network if the interface methods are optimized sufficiently.

The validity tests have shown that the segments we have studied closely perform as expected. The tests were able to locate three errors and two modeling assumptions that were producing unexpected results. The errors have been corrected. The modeling assumptions have been refined; the supply module now treats packets as one-dimensional lines with length and is able to correctly simulate loaders that are attached to two different links. These refinements have improved the performance and correctness of the simulator.

However, our methods are limited to studying a few segments closely, while verifying the correctness of the rest of the simulation at a very aggregate level. To truly test the correctness of the simulation, a comparison between the results of the supply module and MITSIM is necessary. A comparison between the results of the supply module and real traffic data is even more desirable.

Finally, the sensitivity analysis, combined with the timing results, presents a valuable tool to DynaMIT users. This tool will enable users to select the most accurate settings for update and advance intervals, while ensuring that the simulation will run in an acceptable period of time. The timing analysis of the CA-T network shows that simulations with the same number of total iterations, regardless of the length of the update interval, have very similar run times when the output to disk is suppressed. The correlation analysis shows that simulations with smaller update intervals are more closely related to the most accurate simulation (update interval of 15 seconds, advance interval of 15 seconds). These analyses show that users should choose the shortest possible update interval and should adjust the advance interval to meet performance requirements. When the output functionality is in use, the user should be cautioned that shorter update intervals will lead to more output, degrading performance over longer update intervals to some extent. However, optimization of the output functions will make this choice even more attractive in terms of run time performance.

A Appendix: Database Design

Attribute	Data Type, Description
zLinks	
linkID	Long Integer, primary key: id number of link
linkLength	Double, length (meters) of link
zSegments	
segmentID	Long Integer, primary key: id number of segment
parentLinkID	Long Integer, id number of parent link referential integrity constraint: zLinks.linkID
position	Double, position (meters) of segment in parent link
length	Double, length (meters) of segment
freeFlowSpeed	Double, maximum speed (m/s) of packets on segment
jamDensity	Double, jam density (packets/m) of segment
inputFlow	Double, maximum input flow (packets/s) of segment
alpha	Double, alpha parameter of segment
beta	Double, beta parameter of segment
zLaneGroups	
laneGroupID	Long Integer, primary key: id number of lane group
parentSegmentID	Long Integer, id number of parent segment referential integrity constraint: zSegments.segmentID
outputFlow	Double, maximum output flow (packets/s) of lane group
zLanes	
laneID	Long Integer, primary key: id number of lane
parentLaneGroupID	Long Integer, id number of parent lane group referential integrity constraint: zLaneGroups.laneGroupID
zSensors	
sensorID	Long Integer, primary key: id number of sensor
location	Long Integer, segment id where this sensor is located referential integrity constraint: zSegments.segmentID
position	Double, position (meters) of sensor on segment

Table 15: Database Design, Network Topology Connectors

Attribute	Data Type and Description
zNodeUplinks	
nodeID	Long Integer, primary key: id number of node
linkID	Long Integer, primary key: id number of upstream link referential integrity constraint: zLinks.linkID
zNodeDownlinks	
nodeID	Long Integer, primary key: id number of node
linkID	primary key: id number of downstream link referential integrity constraint: zLinks.linkID
zLoaders	
loaderID	Long Integer, primary key: id number of loader
connectionNodeID	Long Integer, node to which this loader is connected should be a member of zNodesDownlinks.nodeID or zNodesUplinks.nodeID
inputFlow	Double, maximum flow (packets/s) into loader
outputFlow	Double, maximum flow (packets/s) out of loader
zPathTopo	
pathID	Long Integer, primary key: id number of this path
linkID	Long Integer, primary key: id number of this link referential integrity constraint: zLinks.linkID
nextPathID	Long Integer, id number of next path from pathID
nextLinkID	Long Integer, id number of next link from pathID and linkID referential integrity constraint: zLinks.linkID

Table 16: Database Design, Network Topology Connections

Attribute	Data Type and Description
zPacketFile	
packetID	Long Integer, primary key: id number of packet
origin	Long Integer, id of loader where packet will be loaded referential integrity constraint: zLoaders.loaderID
destination	Long Integer id of loader where packet will be removed from network referential integrity constraint: zLoaders.loaderID
currentLink	Long Integer, id of link where packet is currently located referential integrity constraint: zPathTopo.linkID
currentPath	Long Integer, id of path that packet is currently on referential integrity constraint: zPathTopo.pathID
nextLink	Long Integer id of next link where packet will be located when loaded referential integrity constraint: zPathTopo.nextLinkID
nextPath	Long Integer, id of path that packet will be loaded onto referential integrity constraint: zPathTopo.nextPathID
location	Long Integer, id of segment that packet is currently on referential integrity constraint: zSegments.segmentID
position	Double, position (meters) where packet is located on segment
length	Double, length (meters) of packet
departureTime	Double, time (seconds) when packet is scheduled to depart
numVehicles	Integer, number of vehicles contained in packet

Table 17: Database Design, Packet File

Attribute	Data Type and Description
density	
timestep	Integer, primary key: update interval number
segmentID	Long Integer, primary key: id number of segment referential integrity constraint: zSegments.segmentID
segmentDensity	Double, density (vehicles/m) of segment
queueLength	
timestep	Integer, primary key: update interval number
laneID	Long Integer, primary key: id number of lane referential integrity constraint: zLanes.laneID
queueLength	Double, length (meters) of queue
flow	
sensorID	Long Integer, primary key: id number of sensor referential integrity constraint: zSensors.sensorID
packetID	Long Integer, primary key: id number of packet referential integrity constraint: zPacketFile.packetID
currentTime	Double, time (seconds) when packet crossed sensor
speed	
sensorID	Long Integer, primary key: id number of sensor referential integrity constraint: zSensors.sensorID
packetID	Long Integer, primary key: id number of packet referential integrity constraint: zPacketFile.packetID
speed	Double, speed recorded (m/s) when packet crossed sensor
travelTime	
packetID	Long Integer, primary key: id number of packet referential integrity constraint: zPacketFile.packetID
currentLinkID	Long Integer primary key: id number of link just traversed by packet referential integrity constraint: zPathTopo.linkID
nextLinkID	Long Integer primary key: id number of link packet has just entered referential integrity constraint: zPathTopo.nextLinkID
startTime	Double time (seconds) when packet arrived at currentLinkID
travelTime	Double time (seconds) it took packet to traverse currentLinkID

Table 18: Database Design, Output Files