Modeling Vehicular Traffic and Mobility for Vehicular Wireless Networks

David R. Choffnes Fabián E. Bustamante
Department of Computer Science
Northwestern University
{drchoffnes,fabianb}@cs.northwestern.edu

Abstract

Ad-hoc wireless communication among highly dynamic, mobile nodes in a urban network is a critical capability for a wide range of important applications including automated vehicles, real-time traffic monitoring, and battleground communication. When evaluating application performance through simulation, a realistic mobility model for vehicular ad-hoc networks (VANETs) is critical for accurate results. This technical report discusses the implementation of STRAW, a new mobility model for VANETs in which nodes move according to a realistic vehicular traffic model on roads defined by real street map data. The challenge is to create a traffic model that accounts for individual vehicle motion without incurring significant overhead relative to the cost of performing the wireless network simulation. We identify essential and optional techniques for modeling vehicular motion that can be integrated into any wireless network simulator. We then detail choices we made in implementing STRAW.

1 Introduction

Communication in mobile ad-hoc wireless networks (MANETs) is the focus of extensive research due to its ability to enable distributed applications among mobile nodes in infrastructureless environments. Vehicular ad-hoc networks (VANETs) are a particularly challenging class of MANETs characterized by nodes with relatively high mobility (speeds between 0 and 20 m/s). In addition, unlike many other mobile ad-hoc environments where node movement occurs in an open field (such as conference rooms and cafés), vehicular nodes are constrained to streets often separated by buildings, trees or other obstructions, thereby increasing the average distance between nodes and, in most cases, reducing the overall signal strength received at each node. Connectivity in this environment is essential for a wide range of important applications including real-time traffic monitoring, battleground communication and other vehicular distributed systems.

We argue that a more realistic mobility model with the appropriate level of detail [9] for VANETs is critical for accurate network simulation results. With this in mind, we designed a new mobility model for VANETs, STRAW (STreet RAndom Waypoint), that constrains node movement to streets defined by map data for real cities and limits their mobility according to vehicular congestion and simplified traffic control mechanisms.

In a separate paper [5], we are evaluating and comparing ad-hoc routing performance for vehicular nodes when using STRAW mobility in diverse urban environments to the performance when nodes move in an open field using the classical random waypoint (RWP) model. Early results indicate that the performance of wireless network protocols in urban environments is dramatically different than that in an open-field/RWP scenario and, further, that the type of urban environment can have a significant impact on the performance of a protocol.

In this paper, we discuss STRAW's design in detail, describe a reference implementation for the SWANS [3] network simulator and detail the performance of SWANS for several interesting cases. The following section motivates the need for car mobility models in ad-hoc networks. In Section 3, we describe the features of a realistic vehicular mobility model. Section 4 details the implementation of STRAW. In Section 5 we discuss and evaluate STRAW's performance and we conclude in Section 6.

2 Background

Routing messages in MANETs has become the focus of much research. Some of the routing protocols that have achieved prominence include topology-based protocols (e.g., DSDV [20], DSR [10], AODV [19] and MRP [18]) that rely exclusively upon IP addresses to locate nodes and location-based protocols (e.g., DREAM [4], GPSR [11]/GLS [14] [17]) that use geographical position for this task.

Proposed protocols are compared against competing or ideal ones in terms of metrics such as packet delivery ratio, throughput, latency and overhead. Due to the prohibitive cost and time constraints of evaluating ad-hoc network protocols in real-world deployments, most studies rely on simulators for experimentation (e.g. [15, 27, 2]).

When analyzing different protocols in simulation, researchers often adopt a common set of configuration parameters, such as:

- Nodes transmit signals that propagate without error to other nodes within a radius of 250 m [13].
- Nodes move in an open field according to a random waypoint model [26] or the Manhattan mobility model [7] with arbitrary pause times and often with arbitrary speed distributions between 0 and 20 m/s.
- The number of nodes is small (i.e., ≤ 100).

Such parameter settings are clearly inadequate for many MANETs, and particularly for VANETs for the following reasons:

- The relationship between distance and signal reception between two nodes is, at best, weakly correlated over large distances [13]. It is also well known that radio transmission range does not form a circle and, for commodity hardware, rarely achieves a 250 m range in common environments.
- Besides settings such as conventions in large conference halls, it is difficult to imagine many scenarios in which nodes move in a open field. Even in such settings, node mobility is not accurately modeled by random waypoints. Specifically in VANETs, nodes must be constrained to roads and adjust their velocities according to traffic control mechanisms, speed limits and the behavior of nearby vehicles. Further, in VANETs, most vehicles attempt to follow paths that minimize trip duration between origin and destination.

- In VANETs, nodes in urban environments can easily number in the thousands or tens of thousands.

Recent interest in VANETs [23] [8] has encouraged researchers to design experiments that better model real vehicular traffic scenarios. For example, [12] studies the behavior of the MAC layer in a vehicular environment using arbitrary road plans while [25] and [24] use the CORSIM traffic microsimulator to provide mobility traces.

A small number of researchers have accounted for street-constrained motion using real road plans in their VANET simulations. In Saha and Johnson [22], the authors state that a random waypoint model is sufficiently similar to the street mobility in terms of network connectivity. The authors reach this conclusion using a 500 m transmission range and an unspecified path loss model. Further, their mobility model does not account for realistic vehicular traffic phenomena such as car-following and traffic control at intersections.

In [24], the authors use CORSIM to provide a highly accurate model of vehicular movement. However, in this case, the vehicular network simulator is detached from the wireless network simulator, making it difficult to close the feedback loop in applications such as "traffic advisory," where participating nodes may alter their routes based on real-time observed traffic conditions. For example, in such an environment, the participating nodes are likely to alter their route to reduce travel time if there is congestion along their current routes. In this case, it is likely that the density of participating nodes along such "faster routes" will be higher than on slower routes, further altering network connectivity by increasing interference.

Many accurate models for simulating vehicular traffic exist, so why build a new model? In wireless network simulators, each node is treated individually for purposes of sending and receiving messages and repositioning the node on a field according to its mobility model. Because wireless network performance and location are tightly coupled, one cannot attain accurate wireless network simulation results unless the underlying mobility model is sufficiently accurate. Unfortunately, in many vehicular traffic simulators vehicles are treated individually only when they enter or leave a segment; when inside a segment, all vehicles are indistinguishable from each other. This critical design choice necessitates an alternate traffic model to ensure accurate wireless network simulation results.

3 Vehicular Mobility Models

We now present vehicular mobility components that can be included in a network simulator. Each component supports variable levels of detail according to the number of parameters that are defined for the simulation. If tuned to empirical data, the parameters can improve simulation accuracy, often at the cost of increased simulation complexity and runtime.

For the purposes of this discussion, we divide the mobility model in our simulator into an intra-segment component, an inter-segment component and a route management and execution component (Fig. 1). We discuss these components in order.

3.1 Intra-segment Mobility

The intra-segment mobility component controls vehicular motion from the point at which a vehicle enters a *road segment*, or *link*, (i.e., a portion of road between two intersections) to the point at which it exits the segment. For this component, we consider only the well-known *car-following model* of vehicular motion. At the simplest level, this model states that a vehicle moves at or near the same speed as the

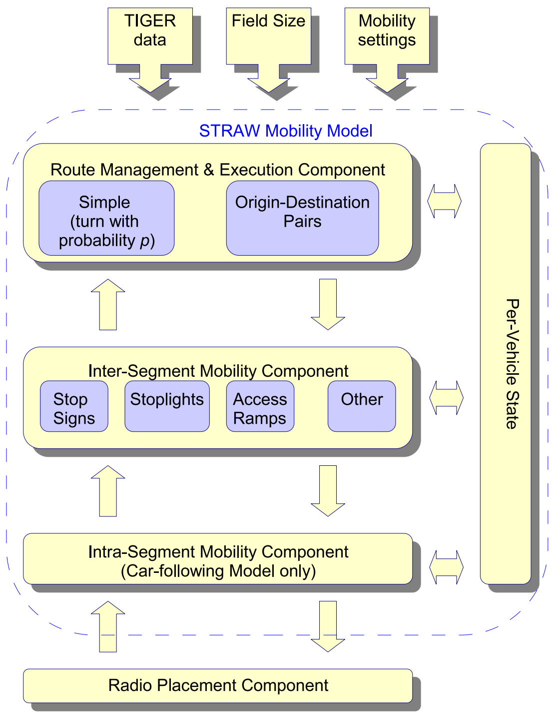


Figure 1: Illustration of vehicular mobility components and their interactions in STRAW.

vehicle in front of it, if there is such a vehicle within sufficient range of the current vehicle. Two important parameters for this model are the speed of the vehicle being followed and the space between the followed and the the following vehicle. There are many ways to determine this intervehicle distance, though it is often modeled as a polynomial function of velocity [21].

The car-following model does not specify a vehicle's behavior when there is no other (nearby) vehicle to follow. We assume that if a vehicle is not within a window of inter-vehicle spacing defined by the car-following model, it accelerates at its specified rate until reaching the vehicle's maximum speed for the current segment. The acceleration rate can be constant, dependent on the current speed or the "type" of driver (e.g., aggressive or defensive driver, hurried or "Sunday" driver). Similarly, a vehicle's maximum speed can be set to the the speed limit of the segment being traversed, a value assigned according to some distribution around that speed limit or a value that is dependent on the aforementioned "type" of driver.

The intra-segment component must also specify how non-following vehicles behave when encountering traffic control. We consider two primary forms of traffic control: stop signs and stoplights. Some forms of traffic control, such as railroad crossing gates, can be generalized to one of these types of traffic control; others, such as yield signs and speed-limit changes must be modeled differently. In the case of stop signs or red stoplights, an approaching vehicle must come to a stop. A yellow stoplight will cause a vehicle to come to a stop only if it cannot cross the intersection before the light turns red. For all cases in which a vehicle must come to a stop, the vehicle must decelerate to zero velocity before encountering the intersection. This can be accomplished with a single, global deceleration rate, a speed-dependent rate or a rate that varies between vehicles according to some distribution.

Another important component of intra-segment mobility is the notion of lane changes. A vehicle can change lanes only if there is space available in an immediately adjacent lane. We consider two reasons for lane changing: increasing speed and preparation for turning.[1] In the former case, if the average speed in an adjacent lane is higher than the current lane, it is likely that the lane change can occur. We contend this is true because a higher average speed indicates not only that the lane has less congestion, but that the

[1] Arguably, a third reason for changing lanes could be described simply as "personal preference," but we choose not to discuss this model as it is difficult to model accurately.

inter-vehicle spacing is greater.

For the purposes of changing lanes to execute a turn, it is quite likely that the turning vehicle will cause the average speed of the current lane to decrease. In fact, in a highly congested network, there may never be enough space to change lanes. To avoid indefinite postponement, it is common for a driver in one lane to allow space for a driver attempting to change to the current lane. One can model this scenario by implementing a "signaling" method that causes a vehicle in the adjacent lane to make room for the incoming vehicle with some probability.

3.2 Inter-segment Mobility

The inter-segment mobility component determines the behavior of vehicles between road segments; i.e., at intersections. The inter-segment mobility component can classify intersections according to the number of intersecting road segments, the types of road segments, and the type of traffic control, if any, at the intersection. In essence, the inter-segment mobility component must perform admission control at each intersection. The traffic-control rules vary according to the intersection type. For the purposes of this discussion, we assume that the Route Management and Execution component discussed in Section 3.3 has already selected the next road segment before the vehicle encounters the intersection and that the vehicle discussed is not currently following another vehicle when it determines the action to take at the intersection.

If there is no traffic control at an intersection, we assume that there is a merging scenario (e.g., from an access ramp onto a highway). In this case, the admission-control mechanism must determine if there is enough space for the incoming vehicle to enter the adjacent lane of the new road segment. If so, the vehicle may enter; otherwise, the vehicle must slow down until space becomes available. Similar to the lane-changing component, this component should include a mechanism to prevent indefinite postponement.

If stop signs are present, the admission control mechanism must consider the number of intersections containing the signs. For instance, if the intersection is an "all-way" stop, a vehicle is admitted into the next road segment only if there is room in the next stop, and only after coming to a complete stop and waiting until its turn to advance. To prevent indefinite postponement, one may assign a total linear ordering to streets in the intersection that determine the order of release from the stopped position. At some intersections, one road segment has a stop sign, while cross traffic does not. In this case, a vehicle at a stop sign can cross the intersection only if moving to the next road segment would not cause a collision with another vehicle (e.g., cross traffic). Note that this condition accounts for the case where a vehicle cannot enter an intersection because the next road segment is already full.

If the intersection uses stoplights for traffic control, the inter-segment mobility component must consider three cases: green, yellow and red lights. Of these colors, there can be more than one type (e.g., a guarded turn signal). When a vehicle approaches an intersection containing a red light, it should begin to slow down at the location where the vehicle's deceleration rate curve would cause the vehicle to stop just before the intersection. Upon encountering a yellow light, the vehicle can cross the intersection only if there is room on the next segment and if the vehicle cannot safely come to a stop before the intersection. Finally, upon encountering a green light, the vehicle may cross the intersection without slowing down, provided that the next road segment is not full. If the light is green and the vehicle executes a turn, the vehicle may proceed only if the next road segment is not full and, in the case of a left turn, there is no oncoming traffic; otherwise, the vehicle must come to a stop at the intersection. Assuming that the vehicle can make make the turn, it must slow down to the maximum turning speed for that vehicle before executing the turn.

3.3 Route Management and Execution

The Route Management and Execution (RME) component determines the ordered set of road segments that a vehicle will traverse during a simulation run. It must ensure that the sequence of road segments along a vehicles path are continuous. The segments along a path can be chosen deterministically, stochastically or a combination of both.

In this paper, we discuss two RME implementations for STRAW. The first is a simple, modified random waypoint model that requires no origin-destination (OD) information. Unlike traditional random waypoint models, this component determines a vehicle's trajectory at each intersection; namely, a vehicle will make a turn at an intersection with a specified probability that can be independently assigned to each vehicle.

The second implementation uses OD pairs and interarrival times to drive the mobility in the network. In this implementation, an OD pair is chosen for each vehicle and routes are initially calculated according to a minimum cost (e.g., fastest time, shortest distance). This implementation can be configured to recalculate a vehicle's route if the cost of a path along or near its precalculated route significantly changes, thus enabling each vehicle to react to traffic information.

Note that both implementations are independent of the underlying vehicular mobility components. We detail the implementation of these mobility components in the next section.

4 Vehicular Traffic Simulation Implementation

In this section, we describe the implementation-specific elements to enable efficient interaction among the various mobility model components. We integrated our mobility model with the JiST/SWANS network simulator. JiST (Java in Simulation Time) is a discrete-event simulator that features high throughput and automatic porting of application code to run in simulation time [2]. SWANS (Scalable Wireless Ad-hoc Network Simulator) is a modular and flexible wireless ad-hoc network simulator that runs atop JiST [3]. Although our implementation is written in Java, it can easily be ported to any language supporting user-defined types. Our vehicular mobility model implementation extends interfaces provided by the SWANS simulator in the `jist.swans.field` package, including the `Field` interface, which encapsulates functionality for mapping radios to locations, the `Mobility` interface, which provides interfaces for implementing the mobility model and the `Spatial` interface, which provides interfaces for locating nodes in the `Field`. The classes that implement our vehicular mobility model are contained in the `jist.swans.field.streets` package.

Before discussing vehicular mobility components, we present some basic concepts particular to our simulation environment. In all of our simulations, individual vehicles are identified by a unique integer value that maps directly to the node id assigned to the vehicle's radio. We have also extended SWANS to incorporate a notion of a penetration ratio; i.e., a percentage of vehicles in the network that are equipped with radios. To enable integration with our network simulator, we represent vehicles without radios simply as vehicles with radios that cannot send or receive. This enables vehicles that are not participating in network communication to interact with all other vehicles.

4.1 Model-independent implementation

This section details the implementation of model-independent components of our vehicular mobility implementation. These components are encapsulated in the `RoadSegment`, `StreetName`, `Shape`, `Intersection` and `SpatialStreets` classes.

Before discussing the detailed implementation of these classes, it is important to describe how map data is loaded into the simulator. This is performed by the `StreetMobility` abstract class, which implements the `Mobility` interface and is extended by the RME components to determine the next road.

Upon initialization, the `StreetMobility` class loads street information from files containing the road segment information, road segment shape and street name. Note that the following relationships hold: each road segment has exactly one street name and zero or one shape. Further, street names may be assigned to one or more road segments, while shapes are assigned to exactly one road segment. If the road segment has no entry in the shape file, the segment forms a straight line; else the points along the road are described by information in the shape file. The road segment, street name and shape data are stored in flat files containing fixed-length records. Thus, each road segment entry contains a pointer to its corresponding shape record (if any) and street name record.

When the `StreetMobility` constructor is called, the user can specify, among other parameters, the latitude-longitude of the bottom-left (Southwest) and top-right (Northeast) corners of the region to which vehicle mobility should be limited. To reduce memory consumption, only road segments that contain both endpoints in the specified region are loaded into the simulator. Similarly, only street names and shapes associated with these road segments are loaded.

After each `RoadSegment` is loaded into the simulator, a reference to that object is placed in a `Vector`. The `RoadSegment Vector` allows fast access to `RoadSegments` identified by its index (an `int`). This is particularly useful, for example, when determining initial vehicle placement using random road segments and for random OD pairs. A reference to each `RoadSegment` is also loaded into a quad tree, or hierarchical grid, containing a `LinkedLists` of `Intersection` objects as leaves. An `Intersection` object contains a `LinkedList` of `RoadSegments`, a location representing its center (in latitude/longitude) and a count of the number of streets. Because map data may be imperfect, a `RoadSegment` is added to an `Intersection` if one of its endpoints is within a user-defined distance (5 m is usually sufficient) from an existing `Intersection`. The `Intersection` class also provides fields and methods to facilitate the implementation of traffic control. Because Java 1.4.x does not include a quad tree implementation, we use the `SpatialStreets` class (an extension of the `Spatial` class provided by SWANS) to maintain the quad tree. The degree of the quad tree can be specified by the user at runtime.

After loading `RoadSegments` and completing the construction of the quad tree of `Intersections`, the `StreetMobility` constructor loads street names and shapes into `StreetName` and `Shape` objects. Because the number of streets and segment shapes actually used in a simulation may vary, but the street and shape indexes are constant for a particular county, `StreetName` and `Shape` objects are placed in `HashMap` objects, where the value of the index is the key and the reference to the object is the value.

The `RoadSegment` class includes the following fields containing information provided by the

USCB's TIGER data files [16][2]:

```
int startAddressLeft;
int endAddressLeft;
int startAddressRight;
int endAddressRight;
Location2D startPoint;
Location2D endPoint;
char roadClass;³
int numberOfLanesToStart;
int numberOfLanesToFinish;
```

Note that the `startPoint` and `endPoint` values are assigned arbitrarily from the `RoadSegments` endpoints, but the values are consistent for the duration of the simulation and are used to determine the trajectory for each vehicle along the segment. Also note that locations are currently represented as two-dimensional points because the TIGER data files do not supply altitude information.

The `RoadSegment` class also contains the index of the street name index, shape index and index in the `Vector` of `RoadSegments` as follows:

```
int streetIndex;
int shapeIndex;
int selfIndex;
```

[2]Note that the Tiger data files do not contain information about whether a road segment is one way. Further, estimates for the number of lanes and the speed limit for a segment are inferred from its road class.

[3]This assists in estimating the speed limit for the segment.

The `RoadSegment` class maintains several properties that aid in vehicle management within and between `RoadSegments`, such as length of the segment, the maximum number of vehicles allowed on a segment, the average vehicle size[4], and following-distance-related constants.

In addition to these constant values, the `RoadSegment` class contains properties for maintaining runtime state about vehicles on each `RoadSegment`. These include the number of vehicles on the road segment, the number of lanes in the segment and a linked list of vehicles for each lane in the segment:

```
int numberOfCars;
LinkedList carsToEnd [];
LinkedList carsToStart [];
```

The remaining classes mentioned in this section are straightforward. The `StreetName` class maintains a set of `Strings` containing the street's prefix (e.g., N, S, E, W), name and suffix (e.g., Ln, Blvd, etc.). The `Shape` class represents a multisegment shape as an array of latitude/longitude pairs.

4.2 Initial Node Placement

Before the simulation can start, vehicles must be placed on valid locations on the road plan for the specified region. Currently, the simulator supports a random placement component, implemented by the `StreetPlacementRandom` class, which extends the SWANS simulator's `Placement` interface. This simple component selects a `RoadSegment` at random, then chooses a direction and a lane at random. To simplify the implementation, the first

[4]Our implementation currently supports only average vehicle lengths but can be extended to support a distribution of vehicle lengths, should the data become available to us.

vehicle in a lane is placed at the front of the lane and subsequent vehicles assigned to that lane are placed behind the last vehicle in the lane. All nodes start with zero velocity. Though simple, this model of placement is sufficiently realistic if vehicles are provided a "warm-up" period during which vehicles move, but no packet traffic is generated. This allows the vehicles to reach cruising speeds and to change streets before network performance is measured. We routinely include a warm-up time of at least 30 seconds in each of our simulation runs.

Future iterations of the node placement component will include support for traffic flows such that vehicles enter and exit the field at various times during the simulation run. This implementation will also include support for incoming flow rates at various locations.

4.3 Intra-segment mobility implementation

In this section, we detail the implementation of our intra-segment mobility component. The implementation consists of the `StreetMobility` class, which implements the `Mobility` interface to provide a node's position after each time step.

When the simulation starts, nodes move according to the *car-following* model such that nodes will attempt to accelerate at a constant rate of up to 5 mph per second to move with a speed equal to the maximum speed for the current driver.[5] This speed is equal to the speed limit for the current road plus a Gaussian distributed value with a zero mean and a (tunable) 4 mph standard deviation.[6] The car will alter its speed according to the following rules:

- *The car encounters an intersection and the next road segment on which it will travel is full.* In this case, the car stops before the intersection and remains stopped until there is room in the next road segment.

- *There is a car in front of the current car.* In this case, the node will slow down to the speed necessary to maintain a speed-based following distance between the current node and the node in front of it. We use the simple formula cited in [21]:

$$S = \alpha + \beta V + \gamma V^2, \tag{1}$$

  where

  α = the vehicle length

  β = the reaction time (we use 0.75 seconds)

  γ = the reciprocal of twice the maximum average deceleration of the following vehicle (we use the empirically-derived value, $0.0070104 s^2/m$ [21])

  If the car in front of the current car is moving faster than the current car, no speed adjustment is necessary.

- *The car encounters traffic control.* In this case, the car will slow down (at a uniform acceleration) before an intersection with a red stoplight or a stop sign; if the stoplight turns green, the car attempts to accelerate if possible.

- *The car turns onto a new street.* In this case, the car slows down before the intersection to make

[5] We acknowledge that acceleration rates are hardly uniform in real life, but this simplifying assumption reduces program and computational complexity. Future iterations of the mobility model will include more accurate acceleration curves.

[6] According to the NHTSA [1], traffic engineers take drivers' perceptions into account in setting speed limits. A posted speed limit is often set to the speed at which 85 percent of drivers travel at or below. However, [6] reports that observed speeds are normally distributed with a center at the posted speed limit. Unfortunately, we could not find a widely accepted mean for this distribution.

the turn at a reasonable speed (5 mph), then accelerates, if possible, to the highest speed it can attain given the other constraints.

Because in our experiments nodes are constrained to roads in downtown urban environments and therefore exhibit average speeds no larger that 12 m/s, we update each node's position once per second using its current speed and direction. We intend to incorporate speed-based position updates in future iterations of STRAW. Finally, it is also important to note that lane changing has not been incorporated into our simulator at this point.

4.4 Inter-segment mobility implementation

This section discusses the implementation of our inter-segment mobility component. Our simulator supports two levels of admission control at an intersection.

The first form of admission control simulates common traffic control mechanisms. Our simulator supports stop signs and timed traffic lights. (Lights for guarded turns are not currently supported.) We expect that future iterations of the component will include triggered lights and guarded turns.

The `Intersection` class provides traffic control functionality in our simulator. In addition to maintaining the location of the center of the intersection, the `Intersection` object also contains other state information, such as the list of `RoadSegments` incident on the intersection, the number and index of unique streets incident on the intersection and the number of streets of each road class for this intersection. This class also contains fields to facilitate the synchronization of nodes attempting to cross an intersection.

The `Intersection` class performs admission control via the `getPauseTime` method, which returns the number of seconds for which a node must pause at the intersection. A nonzero value indicates that a node must stop; a zero value indicates that the vehicle may cross the intersection.

Because real-world, per-intersection traffic control information is unavailable, the simulator currently assigns traffic control according to the class of road segments at each intersection. For example, a stop sign controls traffic between two local/neighborhood roads; access to the intersection between "secondary" roads and state highways is controlled by a timed stoplight. The types of traffic control at various intersections is given in Table 1. Our traffic light implementation currently supports only two streets (up to four road segments) when using street lights. Although the simulation will run if there are more than two streets in such an intersection, it will not correctly ensure that traffic flows without collision.

If the light is red, the `getPauseTime` method returns the number of seconds until the light turns green; otherwise, the `getPauseTime` method returns zero, indicating that the vehicle may cross the intersection.

For simplicity, we used timed stoplights that turn red and green at regular intervals that are dependent on the simulation time. This means that all of the stoplights for intersections of the same type are synchronized, an assumption that is invalid in the real world, in general.

If a vehicle encounters a stop sign, the `getPauseTime` method determines the vehicle's stop time according to the state of the intersection. In the simplest case, if there are no vehicles currently at or waiting to cross the intersection, the vehicle stops for one second and then continues moving. If the `Intersection` object has already selected a vehicle, V_A, to cross the intersection and V_A has not yet done so, a different vehicle on the same street, but on the opposite side of the intersection from V_A, may cross. Otherwise, the vehicle is added to the list of waiting

Road Class	Interstate	US Highway	Secondary	Urban/Rural	Ramp
Interstate	stoplight (30)	stoplight (30)	stoplight(15)	stop sign	no pause
US Highway	stoplight (30)	stoplight (30)	stoplight (15)	stop sign	no pause
Secondary	stoplight (45)	stoplight (45)	stoplight(30)	stop sign	stop sign
Urban/Rural	no pause	no pause	no pause	stop sign	stop sign
Ramp	no pause	no pause	no pause	no pause	no pause

Table 1: Table of traffic control and pause times according to intersecting street types. The column header represents the current street type and the row header represents the intersecting street type. The values in parentheses represents the number of seconds per green light at that intersection.

vehicles and pauses for three seconds (allowing V_A to cross) before it can attempt to cross the intersection by calling `getPauseTime` again.

To prevent indefinite postponement, the `Intersection` object contains a field that specifies the identifier of the next street on which vehicles can cross the intersection. The "next street" is changed after the previous street is serviced; the streets at an intersection are serviced round robin. If there is no contention at an intersection, however, the street with one ore more vehicles is serviced immediately. Although real drivers do not necessarily behave in such a reasonable manner, we believe that this implementation is sufficiently accurate for modeling vehicle interactions at stop-sign intersections.

Another type of admission control is regulated by the capacity of the next road segment on which the vehicle will travel. A node is not allowed to move to the next segment unless there is enough room on that segment. This admission control is performed only after the traffic control admission permits the vehicle to move to the next segment.

The `addNode` method in the `RoadSegment` class performs admission control according to the capacity of lanes in that segment. In the current implementation, this method first finds the lane with the fewest vehicles. If there is room for the incoming vehicle, the method adds the vehicle to the lane and returns a reference to the linked list of vehicles in that lane, for car-following purposes. If there is not room, the method returns `null`. If a vehicle receives `null` from an `addNode` call at an intersection boundary, it remains at the intersection threshold until room becomes available. In particular, the vehicle calls `addNode` every 1/4 second of simulation time until the method returns a valid reference. At this point, the vehicle moves to the next segment on its path, and the intra-segment mobility module manages its motion.

4.5 Per-Vehicle State Information

To manage vehicular mobility efficiently, each vehicle maintains state information in a `StreetMobilityInfo` object. This state object allows the user to configure per-vehicle settings such as its maximum speed, reaction time and acceleration rate, and to maintain information vital to the car-following and inter-segment mobility components, including the road segment that the vehicle is currently on, the direction it is moving, the next road segment it will travel, the vehicle that it is following, the current speed and the remaining distance to the end of the current road segment.

4.6 Route Management and Execution

This section describes the implementations of the Route Management and Execution (RME) component for our STRAW mobility model. We consider two types: simple intersegment mobility and mobility with origin-destination (OD) pairs. In the former implementation, the next segment to which a vehicle will move is determined stochastically at each intersection. In the latter one, the decision is based on the precomputed shortest path between the vehicle's specified origin and destination.

4.6.1 Simple Intersegment Mobility

The simple intersegment mobility implementation maintains a single value to determine the next segment on which a vehicle will travel: the probability that it will turn at any given intersection. This probability can be shared among all vehicles, or can be assigned differently to different vehicles. Although this implementation does not represent any real car-driving phenomenon, it is simple to implement and incurs negligible storage and computation overhead while producing a weak form of random waypoint mobility.[7]

This component is implemented by the `StreetMobilityRandom` class, which extends the `StreetMobility` abstract class by defining the inherited `setNextRoad` method. This method returns the next segment on the same street in the current direction of motion with probability (1-p) and a road segment on a different street (chosen at uniformly at random) with probability p. The value p for a vehicle is maintained by the `StreetMobilityInfoRandom` class, which extends the `StreetMobilityInfo` class.

[7] We describe this motion as "constrained" random waypoint because the set of possible waypoints and the set of possible trajectories are constrained by the fixed street plan. This differs from the random waypoint model in an open field, where waypoints and trajectories are chosen uniformly at random.

4.6.2 Mobility with Origin-Destination Pairs

This scenario models vehicles that move from a start point to an end point along a path that approximately minimizes trip duration according to the speed limit of the available roads. This implementation currently supports three types of motion: a single origin and destination for the duration of the experiment, a sequence of randomly generated origin-destination pairs and a sequence of predetermined origin-destination pairs. In future iterations, we will extend the simulator to support the abundance of existing empirical traffic information that is expressed in flows of vehicles per unit time at a road segment.

When a vehicle is placed on a field and its initial OD pair has been specified, the simulator computes the shortest path between origin and destination. The vehicle then follows the path until reaching the destination. If another OD pair is specified, then the new path is calculated; otherwise, the node is considered to have finished participating in the simulation and is moved off the map (with its radio turned off) to prevent interaction with other nodes.

This component uses the *A* shortest path* algorithm to find a near-optimal shortest path while significantly reducing the range of the problem space explored by using a heuristic function that estimates the distance to the goal. For the purposes of this component, we use the *Manhattan distance* (i.e., sum of the distances along the two orthogonal axes between origin and destination) between the current location and the destination as the heuristic for computing the estimated remaining distance. To reduce the number of turns along a path, and to increase the likelihood of a fast route, the algorithm penalizes turns and non-interstate routes by increasing the costs of paths meeting these criteria.

Mobility with OD pairs is implemented by the `StreetMobilityOD` class, which implements the `StreetMobility` abstract class by defining the `setNextRoad` method, which returns the next road segment along the vehicle's current path. The state for each vehicle is represented by the `StreetMobilityInfoOD` class, which extends the `StreetMobilityInfo` class to include the destination location and the path (a linked list of road segments) from origin to destination.

The A* search is implemented with the `AStarSearch` class, which uses the `SegmentNode` class to represent road segments as nodes in a graph. The `SegmentNode` class implements the `AStarNode` abstract class to provide definitions for the heuristic and cost functions. In the current implementation, the cost of a particular road segment is the estimated speed limit for that segment. In future iterations of this component, we will include other sources for cost analysis, such as current road and traffic conditions.

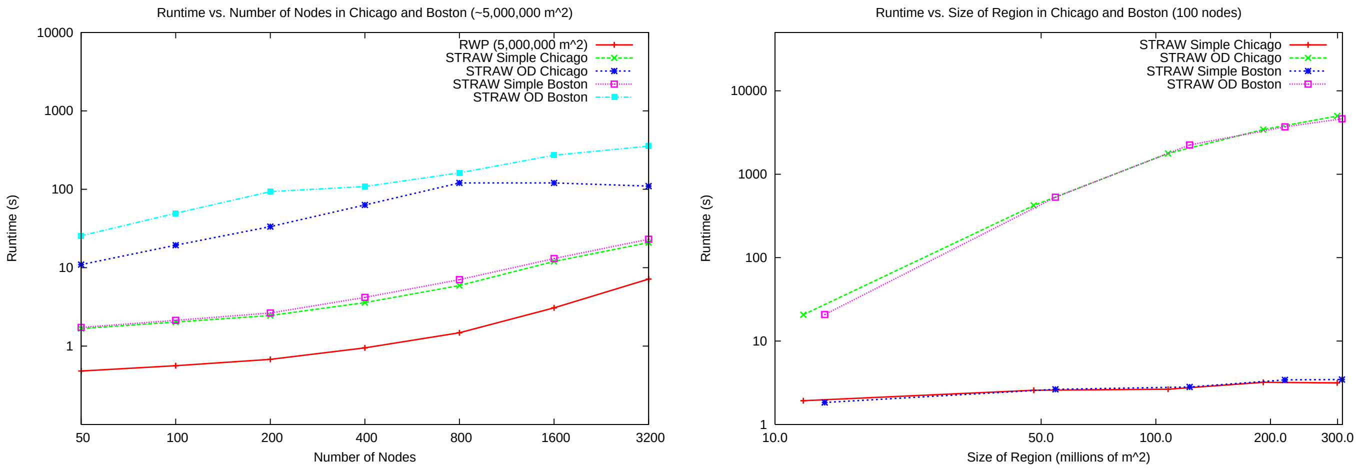
It is important to note that the A* search is by far the most computationally intensive part of our mobility model. In the future, we will implement route caching to improve performance in this area.

5 Performance

In this section, we provide a brief summary of STRAW's performance under JiST/SWANS by evaluating its overhead in comparison to the commonly used open-field random-waypoint mobility mobility model. For all of our figures, we simulated a 16-minute experiment that included a 30-second warm-up time and a 30-second resolution time typical of network performance experiments. Each data point represents the average of five simulation runs, and error bars representing the standard deviation are included if significant. We used the random placement model described in Section 4.2 to determine initial node placement. For STRAW mobility with OD pairs, each time a node reached a destination, we chose a new destination at random and computed the shortest path to that location. These simulations were run on a Linux server equipped with four Intel Xeon 2 GHz processors, though the event dispatcher is single-threaded and used only one of those CPUs.

Figure 2(a) illustrates simulation runtimes according to numbers of nodes in the system. We compare the performance of STRAW in Boston, MA and Chicago, IL to that of the random waypoint model in regions of similar size. As discussed in Section 4.6.1, the "simple" STRAW mobility model incurs a small (approximately constant) factor of runtime overhead compared to the random waypoint model. The STRAW mobility with OD pairs model requires a significantly longer execution time, which is due to the high cost of computing shortest paths, as we discuss later in this section. It is important to note that runtimes for this mobility model eventually decreased as the number of nodes increased in the Chicago region. This occurs because there is significant congestion in the network (i.e., a traffic jam), meaning that each node covers less distance per unit of simulation time and thus will require fewer shortest path searches.

Figure 2(b) illustrates simulation runtimes according to size of the region used in the system. As expected, the "simple" STRAW mobility model incurs a small (approximately constant) amount of runtime overhead regardless of size of the region. The STRAW mobility with OD pairs model is much more sensitive to size, as the shortest path calculation's worst-case execution time is $O\left(b^d\right)$, where b is the number of segments at each intersection and d is the length of the route returned by the algorithm. Thus, as the size of the map grows, the maximum distance between two waypoints increases, which makes d



(a) Effect of number of nodes on runtime for STRAW and a simple random waypoint model (RWP).

(b) Effect of size of region on runtime for STRAW using 100 nodes.

Figure 2: Runtime performance for STRAW when varying the number of nodes and the size of the region.

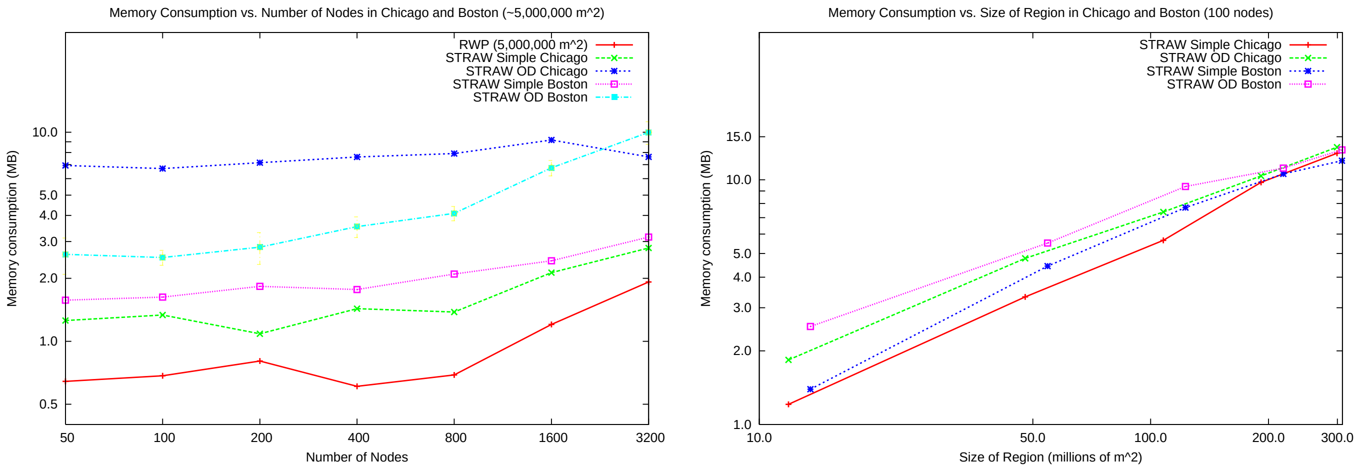
larger. The curves in Fig. 2(b) show that the A* search runtime does not increase exponentially, indicating that the Manhattan distance heuristic improve the scalability of the shortest path search.

Figure 3(a) demonstrates how the simulation's memory consumption varies according to the number of nodes in the system.[8] We include the same mobility models as in the previous figure. In this case, the random waypoint model, which does not load any map data, provides a baseline for the memory consumption in STRAW. Notice that the number of nodes in the system has much smaller effect on memory consumption than it does on runtime. This indicates that memory is not a significant factor when scaling the system to large numbers of nodes.

Figure 3(b) shows that the most significant factor for memory consumption is the number of road segments in the test region, which is directly correlated to the size of the region. The curves indicate that there is a small amount of memory overhead incurred to perform the shortest path calculations, but that memory consumption for both STRAW mobility models is directly proportional to the size of the region. In fact, when loading map data for the entire city of Chicago (230 square miles containing 157,120 road segments, not shown), memory consumption was approximately 92 MB. Although the size of the data structures supporting STRAW varies during the execution, the 92 MB value yields approximately 58 bytes of memory per `RoadSegment`, on average.

The results of our experiments demonstrate that, in general, one can successfully model large-scale

[8] Note that we use the Java API's memory reporting functions to determine memory consumption. Due to Java's garbage collection implementation, it is difficult to determine how much allocated memory is actually being consumed, though the numbers represent an upper bound to memory consumption. We attribute anomalies in the memory consumption graph to this property, not to any intrinsic properties of STRAW or the SWANS simulator.



(a) Effect of number of nodes on memory consumption for STRAW and a simple random waypoint model (RWP).

(b) Effect of size of region on memory consumption for STRAW.

Figure 3: Memory performance for STRAW when varying the number of nodes and the size of the region.

realistic vehicular motion on commodity hardware. Although STRAW with OD pairs does not scale as well as other mobility models, its worst-case performance is bounded by the finite capacity of the underlying road plan.

6 Conclusion and Future Work

This paper described the design principles and one particular implementation of a realistic vehicular mobility model for use in a wireless network simulator. We discussed the motivation for including a realistic mobility model for correctly evaluating the performance of vehicular ad-hoc networks. Then we identified implementation-independent features of vehicular mobility models and proposed a functional decomposition of vehicular mobility models into three components: intra-segment mobility, inter-segment mobility and route management and execution. In this manner, each component can be developed and enhanced independently to improve realism.

We detailed our implementation of the STRAW (STreet RAndom Waypoint) vehicular mobility model and its supporting components, such as the street placement model, the car-following intra-segment mobility implementation, basic traffic control implementations and the route management and execution implementations. Based on this reference implementation, we demonstrated that STRAW mobility provides reasonable runtimes and memory consumption that scales fairly well with the size of the simulation.

The described model is a significant improvement over the random waypoint model and other similar vehicular mobility models. There are, nonetheless, several important details that may further improve the realism of the mobility model. For example, most empirical traffic data concerns traffic flows; i.e., counts of vehicles entering (and/or exiting) a road segment per unit time. Another important aspect of vehicular motion is lane changing. In the future, we will implement flows of traffic, lane changing and we will ensure that vehicles are located in the correct lane before turning at an intersection, for example. We are also interested in implementing the capability to calculate the shortest path between origin and destination by including the current average vehicle speed on a segment to determine the cost of a segment. Finally, we will continue to improve STRAW's memory and runtime performance.

References

[1] *Speed Management Work Plan*, 1997.

[2] BARR, R. *An efficient, unifying approach to simulation using virtual machines*. PhD thesis, Cornell University, 2004.

[3] BARR, R. Swans - scalable wireless ad hoc network simulator. April 2004.

[4] BASAGNI, S., CHLAMTAC, I., SYROTIUK, V. R., AND WOODWARD, B. A. A distance routing effect algorithm for mobility (dream). In *Proc. of ACM/IEEE MobiCom* (1998).

[5] CHOFFNES, D. R., AND BUSTAMANTE, F. E. An integrated mobility and traffic model for vehicular wireless networks. *In preparation* (September 2005).

[6] CLARKE, R. The distribution of deviance and exceeding the speed limit. *Br J Criminol 36*, 2 (1996), 169–181.

[7] F. BAI, N. SADAGOPAN, A. H. Important: A framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks. In *Proc. of IEEE INFOCOM* (2003).

[8] GUIZZO, E. Network of traffic spies built into cars in atlanta. *IEEE Spectrum* (April 2004).

[9] HEIDEMANN, J., BULUSU, N., ELSON, J., INTANAGONWIWAT, C., CHAN LAN, K., XU, Y., YE, W., ESTRIN, D., AND GOVINDAN, R. Effects of detail in wireless network simulation. In *Proc. of SCS Multiconference* (2001).

[10] JOHNSON, D. B., AND MALTZ, D. A. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, Imielinski and Korth, Eds., vol. 353. Kluwer Academic Publishers, 1996.

[11] KARP, B., AND KUNG, H. T. Gpsr: greedy perimeter stateless routing for wireless networks. In *Proc. of ACM/IEEE MobiCom* (2000).

[12] KORKMAZ, G., EKICI, E., ÖZGÜNER, F., AND ÖZGÜNER, U. Urban multi-hop broadcast protocol for inter-vehicle communication systems. In *Proc. of ACM VANET* (2004).

[13] KOTZ, D., NEWPORT, C., GRAY, R. S., LIU, J., YUAN, Y., AND ELLIOTT, C. Experimental evaluation of wireless simulation assumptions. In *Proc. of ACM MSWIM* (2004).

[14] LI, J., JANNOTTI, J., COUTO, D. S. J. D., KARGER, D. R., AND MORRIS, R. A scalable location service for geographic ad hoc routing. In *Proc. of ACM/IEEE MobiCom* (2000).

[15] MCCANNE, S., AND FLOYD, S. ns Network Simulator. http://www.isi.edu/nsnam/ns/.

[16] MILLER, C. L. UA census 2000 TIGER/Line files technical documentation, April 2002.

[17] MORRIS, R., JANNOTTI, J., KAASHOEK, F., LI, J., AND DECOUTO, D. Carnet: a scalable ad hoc wireless network system. In *Proc. of ACM SIGOPS European Workshop* (2000).

[18] NAIN, D., PETIGARA, N., AND BALAKRISHNAN, H. Integrated routing and storage for messaging applications in mobile ad hoc networks. *Mob. Netw. Appl. 9*, 6 (2004), 595–604.

[19] PERKINS, C. Ad hoc on demand distance vector (aodv) routing, 1997.

[20] PERKINS, C., AND BHAGWAT, P. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. of ACM SIGCOMM* (1994).

[21] ROTHERY, R. W. Car following models. In *Trac Flow Theory* (1992), Transportation Research Board, Special Report 165.

[22] SAHA, A. K., AND JOHNSON, D. B. Modeling mobility for vehicular ad-hoc networks. In *Proc. of ACM VANET* (2004).

[23] SMITH, T. Auto makers to create a car-to-car wlan by 2006. The Register (www.theregister.co.uk), December 2004.

[24] WU, H., FUJIMOTO, R., GUENSLER, R., AND HUNTER, M. Mddv: a mobility-centric data dissemination algorithm for vehicular networks. In *Proc. of ACM VANET* (2004).

[25] YIN, J., ELBATT, T., YEUNG, G., RYU, B., HABERMAS, S., KRISHNAN, H., AND TALTY, T. Performance evaluation of safety applications over dsrc vehicular ad hoc networks. In *Proc. of ACM VANET* (2004).

[26] YOON, J., LIU, M., AND NOBLE, B. Sound mobility models. In *Proc. of ACM/IEEE MobiCom* (2003).

[27] ZENG, X., BAGRODIA, R., AND GERLA, M. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Proc. of PADS* (1998).