ABSTRACT

Title of Document: A simulation framework for traffic information dissemination in ubiquitous vehicular ad hoc networks

Hyoungsoo Kim, Doctor of Philosophy, and 2007

Directed By: Associate Professor David J. Lovell, and Department of Civil and Environmental Engineering

The ongoing efforts to apply advanced technologies to help solve transportation problems advanced the growing trend of integrating mobile wireless communications into transportation systems. In particular, vehicular ad hoc networks (VANETs) allow vehicles to constitute a decentralized traffic information system on roadways and to share their own information. This research focused on the development of an integrated transportation and communication simulation framework to build a more realistic environment with which to study VANETs, as compared to previous studies. This research implemented a VANET-based information model into an integrated transportation and communication simulation framework in which these independent simulation tools were tightly coupled and finely synchronized. A traffic information system as a VANET application was built and demonstrated based on the simulation framework developed in this research. In this system, vehicles record their own travel time data, share these data via an ad hoc network, and reroute at split sections

based on stored travel time data. Disseminated speeds of traffic information via broadcast on a real roadway network were obtained. In this research, Traffic information speeds were approximately between the road speed limit in a low traffic density - in which case they were mostly delivered by vehicles traveling on the opposite directions - and half of the transmission range (250/2 meter) per second in a high traffic density, which means they were delivered by vehicles traveling in the same direction. Successful dynamic routing based on stored travel time data was demonstrated with and without an incident in this framework. At the both cases, the benefits from dynamic routing were shown even in the low market penetration. It is believed that a wide range of VANET applications can be designed and assessed using methodologies influenced by and contributed to by the simulation framework and other methods developed in this dissertation.

A SIMULATION FRAMEWORK FOR TRAFFIC INFORMATION DISSEMINATION IN UBIQUITOUS VEHICULAR AD HOC NETWORKS

By

Hyoungsoo Kim

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:

Associate Professor	David J. Lovell, Chair
Professor	Gang-Len Chang
Associate Professor	Richard J. La
Professor	Sung W. Lee
Fellow	Karl E. Wunderlich, Noblis

Acknowledgements

Sincere appreciation is extended to my advisor, Professor David J. Lovell for supervising this dissertation; his time, energy and support were invaluable. He was my academic advisor and, sometimes, an American friend. His patient guidance and inspiring advice help me through many difficult moments. Very special thanks to my dissertation committee members, Dr. Gang-Len Chang, Dr. Richard J. La, Dr. Sung W. Lee, and Dr. Karl E. Wunderlich, for their invaluable advices and interests in my work. I also thank Dr. Paul Schonfeld for always supporting students.

I thank Quadstone Paramics for supporting Paramics software and Scalable Network Technologies for supporting QualNet software for my research. I cannot imagine how I could finish my dissertation without their supports.

During my entire period to study, I have been able to meet many civil engineering colleagues from various countries and discuss not only academic issues but also many other issues with them. It was lucky to me that I met them, and I thank them for being my friends.

Korean friends in our department are more than friends. When I started my study, they were with me, and when I finish my study, they are still with me. They are like brothers and sisters to me. I always feel I am in a big family with them. I thank them for helping and supporting me to study.

It was a great time that I was in the Kostam (Korean Student Tennis Associate at the University of Maryland). The Kostam made me healthy mentally as well as physically. I always waited for the Kostam meeting on Fridays, and I was recharged at that meeting. I met many wonderful friends through the Kostam, who would be

my fortunes in my life. I thank the Kostam for giving me the chance to meet them, and I love the Kostamians.

I obtained two friends at the end of my study: Dr. Minho Shin and Dr. Beomseok Nam. Without them, I could not have finished my dissertation. It was a great time to discuss and work with them. I thank them for their enthusiastic supports.

Sometimes, I think about "family." Unfortunately, there is no word in the world that I can describe family's love, endless, unconditional, and unchangeable love. Still not enough to verbalize… This dissertation is dedicated to my family that I always love.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

This dissertation focuses on a traffic information system based on ad hoc networks. This chapter introduces ad hoc networks as a novel approach to improve traffic mobility and safety, and describes the limitations of previous studies as the background of the dissertation. The objectives of this research and a brief description of the remaining chapters of the dissertation follow the background.

1.1 Background

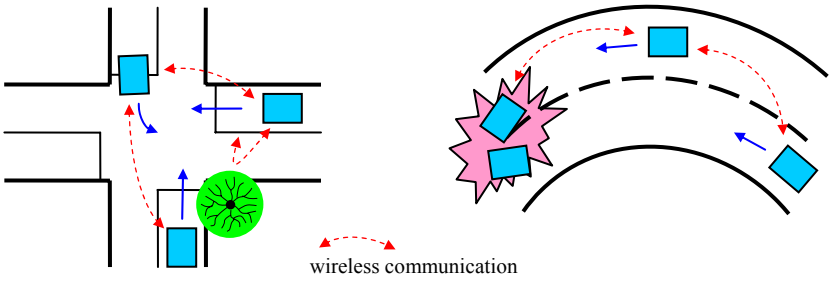
Many efforts have been made to mitigate traffic congestion and accidents by applying advanced technologies to transportation systems. Since the early 1990s, the U.S. Department of Transportation (US DOT) has conducted the Intelligent Transportation Systems (ITS) program in order to improve transportation safety, relieve traffic congestion and enhance infrastructure productivity. Intelligent Transportation Systems encompass a variety of advanced electronics technologies such as communications, sensing, and control (US DOT, 2007).

One novel approach to improve transportation systems is to take advantage of wireless communication technology such as ad hoc networks[1]. An ad hoc network is defined as a collection of devices (nodes) that wish to communicate, but that have no fixed infrastructure available. They have no pre–determined organization of available links (Ramanathan and Redi, 2002). If this technology is, in particular, advanced into

[1] It should be noted that the commonly accepted usage of the phrase "ad hoc network" originates from the colloquial interpretation of ad hoc as meaning without formal organization. The phrase "ad hoc" really means "for a specific purpose" which is not the idea captured by ad hoc networks.

the traveling vehicle fleet, vehicles on roadways would create a mobile ad hoc network which would then enable traveling vehicles to communicate with each other about surrounding traffic states through inter-vehicle communication; in this context, the system is called Vehicular Ad hoc Networks (VANETs). Traffic states (information) shared could be travel time and speed, accident locations, unexpected weather, obstacle warning, and emergency notification. The recent interest in VANETs has led to a flurry of application ideas for transportation systems.

One specific application area of VANETs is improving transportation safety. The general idea of VANET's safety applications is to extend the range of vehicles' "awareness" in curve sections or in intersection areas. Under such schemes, vehicles can learn about dangerous situations ahead of time and slow down to avoid collision. Figure 1-1 shows simple examples of collision avoidance schemes (FleetNet, 2002).



a) Intersection collision warning b) Rear-end collision warning

FIGURE 1-1 Traffic safety application

The composition of collision avoidance system on vehicular ad hoc networks does not differ significantly among different groups conducting that research. In each

case, a vehicle's location from GPS (Global Positioning System) is broadcast on a shared wireless channel at potential accident areas such as intersections and curved sections. All vehicles equipped with communication devices in the vicinity or the original message can then determine the location of the source vehicle (Avila *et al*., 2005, Chisalita and Shahmehri, 2002, Dogan *et al*., 2004, Ueki *et al*. 2004, Xu *et al*., 2004, and Yin *et al*., 2004).

Figure 1-2 illustrates traffic information dissemination, another application of VANETs. Vehicles on roadways create a VANET and communicate with each other about traffic states so that vehicles recognize traffic situations around them. Each vehicle records its own travel experiences over various links, and transmits its experiences to other vehicles so that they can develop an overall understanding of the congestion picture (Chen *et al*., 2001, Bogenberger and Kosch, 2002, Blum *et al*., 2004, Hasegawa *et al*., 2004, Little and Agarwal, 2005, Liu *et al*., 2005, Chawathe, 2006, and Leung *et al*., 2006).

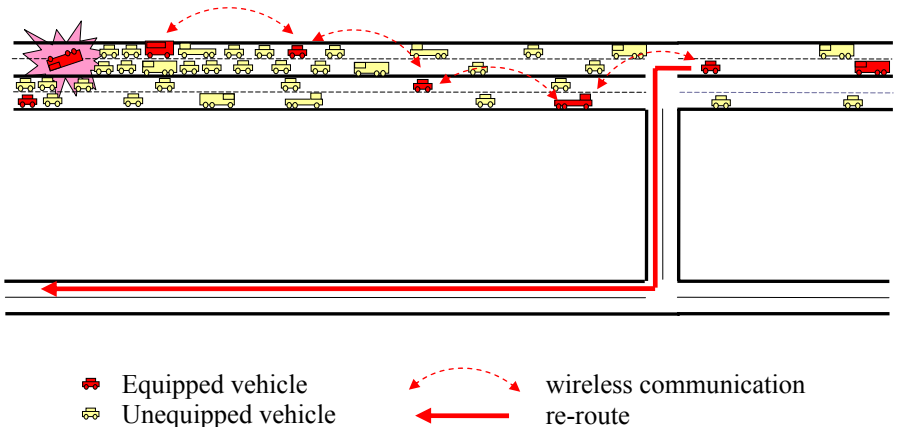


FIGURE 1-2 Traffic information dissemination

Because a working system has not been deployed, it is hard to conduct real experiments for VANET research due to its high cost in time and expenses compared to computer simulation. Although computer simulation is popular for its low cost in both time and expenses, it still confronts a major challenge in terms of reliability, which in this context means the degree to which it replicates the real system it is intended as a proxy for. Simulation of VANET-based transportation systems requires realistic microscopic models of both a transportation system and a wireless communication network.

In order to simulate practical vehicle movements such as car following, lane changing, and shock waves, numerous microscopic traffic simulators have been developed and employed (Yang, 2003, Goel *et al*., 2004, Wischhof *et al*., 2005, Wu, 2005, Xu and Barth, 2006, and Saito *et al*., 2007). Corsim (Corsim homepage, 2007), Paramics (Paramics homepage, 2007), and VISSIM (VISSIM homepage, 2007) are well-known traffic simulation tools that include the logical models mentioned above. These transportation-oriented tools replicate various traffic circumstances as vehicles move on real roadways. In wireless communications simulation, wireless characteristics such as path loss, fading, interference, and communication collision should be considered. These are related to the physical layer and the Medium Access Control (MAC) layer of the Open Systems Interconnection (OSI) reference model which the International Organization for Standardization (ISO) proposed regarding a unique set of functions and responsibilities in order to standardize the protocols used in the various communication network layers.

To date, however, no single simulator alone can simulate a VANET-based transportation system. Most prior works on VANETs ignored one or the other part of the simulations (Goel *et al*., 2004, Wischhof *et al*., 2005, Yang, 2003) or attempted to combine two separate simulators, one for transportation and the other for communication (Saito *et al*., 2007, Wu, 2005, and Xu and Barth, 2006). Prior attempts to combine two simulators have critical limitations on the practical complexity of experiments which can be conducted, particularly in applications such as dynamic routing, and these problems should be overcome for a viable simulation experiment. That is one of the practical contributions of this dissertation.

1.2 Research objectives

This research focuses on a VANET-based traffic information system. A major goal of this research is to develop a simulation framework for VANET-based traffic information systems in which a transportation simulator and a communication simulator are tightly coupled and finely synchronized. In order to achieve this goal, the following objectives will be pursued:

- Review state-of-the-art research related to VANETs. In particular, we focus more on simulation efforts for VANET-based traffic information systems and clarify the critical limitations of previous research (Chapter 2).
- Investigate the characteristics of traffic information which vehicles should treat in VANET-based traffic information systems. Individual and aggregated

traffic information, and temporal and spatial relevance of traffic information are discussed (Chapter 3).

- Design and implement a simulation framework for transportation systems based on VANETs. It is designed in accordance with the information model developed in Chapter 4. It models how information from a transportation system is collected and shared, and how vehicles might react to the data, depending on the application in question. Those results are fed back into the transportation system. In the implementation, a transportation simulator and a communication simulator are tightly coupled and finely synchronized (Chapter 4).
- Build and evaluate the simulation framework for a VANET-based traffic information system. The performance for the framework and the traffic information system are discussed. Case studies are used both to highlight the properties of the conjoined simulation modeling tool, as well as to illustrate certain system aspects of the information that can be collected in this manner. In particular, we highlight how the new simulation environment produces superior results for some questions that have been asked and answered elsewhere in the literature (Chapter 5).

1.3 Dissertation organization

This dissertation is organized as follows. Chapter 2 contains a review of a variety of results from the literature on ad hoc networks and vehicular ad hoc networks. To contrast with the goals of this dissertation, particular attention is paid to traffic

information systems based on a simulation framework. Chapter 3 discusses details of the traffic information which vehicles should collect and share in a traffic information system based on a VANET. Individual traffic information is compared with aggregated information as a proxy of a traffic condition, and temporal and spatial relevance of traffic information are described. In Chapter 4, the simulation framework is designed and implemented. Based on the simulation framework designed, a transportation simulator and a communication simulator used in this research are introduced, and it is described how these simulators are synchronized in terms of time and mobility. As a case study, Chapter 5 shows the application of the simulation framework implemented for a traffic information system. Simulation experiments are designed and evaluated in order to examine the performance for this simulation framework. Chapter 6 concludes the entire research.

Chapter 2: Related Work

This chapter contains a review of related research, and provides some background material on networking protocols, particularly those employed for the latest incarnations of mobile ad hoc networks. The section begins with a review of ad hoc networks in general, followed by specific applications of ad hoc networks for vehicular populations. The section concludes with a set of findings from the existing literature that highlight the context and importance of the studies for this dissertation.

2.1 Ad hoc networks

This section describes some important milestones in the development of ad hoc networks, and gives some technical details about the protocols involved. In part, this is intended to be simply informative. It should be noted, however, that the traffic applications proposed for this research have data characteristics that differ from traditional information being disseminated in ad hoc networks in very important ways, and these suggest some modifications to protocols that have the potential to make vehicular ad hoc networks very efficient and useful for traffic modeling purposes.

2.1.1 The ALOHA network

It is widely acknowledged that the first wireless ad hoc network was the ALOHA system developed in the late 1960's and early 1970's in Hawaii under the leadership of Norman Abramson (see for example Abramson, 1985). The purpose at the time

was to connect computers in academic buildings on the various Hawaiian Islands without using the existing telephone network. Some of the most important arguments for ad hoc networks in certain situations were developed as part of this project. For this reason, the ALOHA system is described in the following paragraphs in some detail.

The ALOHA researchers recognized that networks whose nodes wanted to communicate only randomly and intermittently would not be well–served by highly regulated multiple access protocols such as frequency division (FDMA) or time division (TDMA). The deterministic resource allocation schemes inherent in these protocols meant that significant communication opportunities would be wasted a lot of the time, particularly if communication loads were skewed across nodes. Instead, the ALOHA system pioneered what would eventually become known as "packet communication" (the popularity of the term is due in part to the dissertation by Metcalfe, 1973, who went on to be a co–inventor of Ethernet).

With ALOHA, any node with information to transmit breaks it up into packets and transmits these individually. Each packet concludes with a checksum, so the receiving node can tell (with very high but not perfect reliability) if the information has been corrupted. In order for any single node not to monopolize the carrier, rules must be in place to restrict the length of packets and to enforce a minimum "quiet time" between packets for any individual node.

2.1.2 CSMA/CD, IEEE 802.3, and the Ethernet

Inherent in the ALOHA scheme is the notion that any station that wishes to communicate does so when it desires, at the risk of doing so coincidentally with other stations. If multiple entities try to transmit simultaneously, all of their communications will be garbled. This can be detected with very high probability at the receiving end using checksums, but it also wastes time because nodes continue to transmit, oblivious to the fact that they are being corrupted. An improvement involves the utilization of a transceiver to "listen" to the channel first to make sure it isn't obviously busy, and if not, then try to send its packet – this is called "carrier sense multiple access" (CSMA). Even with this scheme, however, two nodes that attempt to begin communicating almost simultaneously (within the small time window of propagation delay between them) would both perceive an idle network when they first listened, and as a result would begin transmitting, but their transmitted packets would "collide," and be garbled and therefore useless.

Each station must be able to detect these collisions, and then decide if and when to re–transmit its packet. There are several popular methods of collision detection – the combination of CSMA with any collision detection scheme is denoted CSMA/CD. In real time, it is possible to monitor power levels and/or pulse widths with a receiver, at the same time that transmission is taking place. If a node detects a significant difference between what it knows it is transmitting itself and what is being received, then it can conclude that some collision has taken place. This method is most reliable on wired networks where the confounding effects of other interference are not present to anywhere near the same degree as in wireless communications, thus

the ability to distinguish power levels is quite high. Another method, which would be more successful in a wireless environment, is to use "acknowledgement": a central hub replies to any successfully received packet with an acknowledgement message, or in the extreme, a copy of the original packet. If a transmitter receives its own packet back, then it knows it was transmitted successfully, and it can then begin processing its next packet. Thus, in the extreme, each successful transmission requires two nearly identical packet transmissions. There is some chance that the original packet was transmitted successfully, but the acknowledgement message collided, in which case the sender would think the original message was unsuccessful.

The question of when and if to re-transmit, in the event of a collision, is very important. Obviously, this choice should not be made identically across all stations, since this would almost guarantee indefinite packet collisions. Typically, a node that detects a collision first terminates transmission immediately, so as not to waste any time. In some systems, it also broadcasts a brief jamming signal. Since it has decided that it was collided with, it is safe to assume that anyone else currently transmitting will also be affected, so it is better to send a jamming signal that makes that point known very clearly and immediately to all affected nodes. All such nodes are now in a state called "contention": they all have packets they would like to transmit, but they also recognize that other nodes are in the same situation. An individual node now waits for a random period called the "back-off" time and then tries again. In some systems, there is a known prioritization scheme, either for the nodes themselves or for certain message types, and this could influence the relative urgency of the retrial. In some systems, this process of re-trying packets can continue

a number of times, but the back-off time doubles for each collision, resulting in what is known as an "exponential back-off" scheme. After some number of failed attempts, the packet is "dropped." This is very important in applications such as file transfer, because arguably each packet is very important. It might contain a chunk of data or code that is part of a larger file, and thus it is critical for it to be communicated accurately.

The likelihood of collisions (which increases with the number of stations and the volume of traffic on each), plus the stochastic nature with which re–transmissions are made, lends an element of randomness to the delivery of information via such a protocol, and it should be emphasized that such a system is only appropriate for applications where this is not problematic. Applications that require very deterministic behavior tend to use token–based systems instead. The ALOHAnet is capable of a maximum of 18% efficiency (i.e., only 18% of the time useful packets are being transmitted) with multiple competing nodes, before the incidence of packet collisions actually causes the system to degrade.

There is a modified version of ALOHA called slotted ALOHA. In this scheme, time is treated not as a continuum, but rather as a sequence of discrete intervals, each long enough for the transmission of a single packet. Nodes only attempt to initiate transmission at the beginning of one of these intervals. This greatly reduces the period during which a packet is vulnerable to collision from other nodes, and therefore increases the probability of a successful transmission. The throughput efficiency accomplished with slotted ALOHA is about 37 percent, compared to 18 percent for pure ALOHA. This is not the same idea as TDMA, however, since

individual nodes are not assigned to subsets of the slot sequence; any node can use any slot provided it is free.

Perhaps the most ubiquitous protocol now in service is Ethernet, also known in the standards literature as IEEE 802.3. This standard includes specifications for the physical layer, which has details about the kinds of cabling and connections required (see for example Murthy and Manoj, 2004); and the Medium Access Control (MAC) sublayer, which defines a CSMA/CD scheme, including such things as packet length and construction, and protocol details such as back-off times. The IEEE 802.3 includes a minimum frame length, which is helpful because it is longer than what would ordinarily be transmittable within the vulnerability period of an individual node. Thus, packets that are terminated early because of collision are easily distinguishable from full-length, valid packets. Again, these are protocol details that could be adjusted for a network serving exclusively data of relatively low individual importance. The IEEE 802.3 standard also requires each packet to include a destination and source address. This is an important distinction, because there are arguments that can be applied to the vehicular applications in this dissertation that such specificity is not beneficial in all situations, in which case some efficiency can be gained by reducing packet sizes. Ethernet has the built-in ability to send to all nodes (i.e., to "broadcast") by setting all destination address bits to one.

2.1.3 CSMA/CA, IEEE 802.11, and Wi-Fi

Within the set of specifically wireless protocols, the most well known is the IEEE 802.11 family. The purpose of the IEEE 802.11 specification, essentially, is to

translate the success of wired ad hoc protocols such as IEEE 802.3 into the wireless domain, taking account of the particular issues that arise therein. The standard recognizes that the nodes are mobile and unpredictable, and that they come and go with abandon. The IEEE 802.11 physical layer for radio-based networks (there is also a specification for infrared) includes specification of various spread-spectrum frequency allocation mechanisms, within various bands, including 5 GHz, 5.9 GHz (for Dedicated Short Range Communications or DSRC), and the unlicensed 2.4 GHz ISM (Industrial/Scientific /Medical) band. The latter frequency band is available worldwide, and hence is very popular for internet applications and other civilian uses. The 802.11b task group defined the necessary details for the 2.4 GHz band, and this set of specifications is now colloquially known as Wi-Fi, which stands for "wireless fidelity." Table 2-1 shows some details on these characteristics for various members of the IEEE 802.11 family (Werner, 2005 and Liu *et al*., 2005), as well as some closely related protocols not in the 802.11 family.

TABLE 2-1 Near-Range wireless data communication standards

standard	Transmission Rate [Mbps]*	Range [m]	Frequency [GHz]
802.15 (Bluetooth)	1	10	2.4-2.497
802.15.4 (ZigBee)	0.25	30	2.4
802.11b (WiFi)	11	100-200	2.40-2.497
802.11a (WiFi)	54	30-200	5.13-5.35 5.72-5.87
802.11g (WiFi)	54	100-300	2.4
802.11n (draft)**	600	600***	2.4 and 5
802.11p (DSRC)	27 (54)****	1000	5.85-5.925

* This is the maximum data transfer rate that can be supported by a single node maximizing the channel utilization, with no packet collisions. Practical data rates tend to be lower, because collision avoidance, back-off, and packet collisions have a deleterious effect on transmission rate.

** The 802.11n standard is only draft, but is considered stable enough that commercial devices based on the standard are even now widely available. The standard is expected to be ratified in 2007 (Broadcom, 2006).

*** The possible range for 802.11n is colloquially stated as double that of 802.11g.

**** This is the DSRC band. A total of 75 MHz of bandwidth will be divided into 7 smaller bands of 10 MHz each, which will serve different purposes. One or two of these bands might be available for any particular purpose, such as safety (collision avoidance) or traffic information. Each can support a transmission rate of 27 Mbps.

As mentioned above, collision detection is much more difficult with wireless networks than with wired networks. Furthermore, the inherent "noisiness" of the wireless medium causes bit errors much more often than with wired channels (e.g., on average one bit in every 10,000 is in error in a wireless channel, whereas the rate for fiber optic cables might be one in every 1 billion bits). These facts conspire to make collisions much more problematic in wireless channels, to the extent that the design

philosophy is different as they are concerned – rather than simply detect collisions and re-transmit packets if they occur, the protocol is specifically designed to make every effort to avoid collisions in the first place. Generically, this task is called "collision avoidance" (CA) and schemes such as IEEE 802.11 then fall under the moniker CSMA/CA.

Under IEEE 802.11, time is again discretized into slots. Carrier sensing is accomplished (functionally) similar to 802.3, although with some differences in detail. Furthermore, no node can gain immediate access to a channel – each has to wait at a minimum one DCF inter-frame spacing (DIFS), where DCF stands for Distributed Coordination Function, which is the primary access method for this protocol. When DCF is invoked, it is assumed that no fixed access point (AP) is available to mediate medium contention; thus the nodes have to do it themselves. If the channel is busy, the back-off process is initiated. Even during the back-off, any instance of a busy channel causes the back-off counter to be suspended. The back-off time can be reduced for nodes that have been waiting longer – this gives them, essentially, a form of priority over more recent service requests.

Because of the essential nature of the data that is assumed for most applications, it is important that nodes be able to sense if their transmissions were delivered successfully. This is accomplished via an acknowledgement message, as described previously. It is possible to experience a problem known as the "hidden terminal problem," whereby one node can communicate with two other nodes, but these two cannot sense each other, presumably because of distance or perhaps line-of-sight interference. In this case, each of the two nodes might think that they have exclusive

access to the common receiving node, and if they both transmit accordingly, their messages will collide at the receiver. The mechanism to avoid this is based on the time-honored request-to-send-clear-to-send (RTS-CTS) mechanism inherent in older serial communications schemes such as RS-232. In essence, a transmitting node pre-notifies the recipient of an imminent transmission, using an RTS message. If the receiver is ready, it signals its readiness to the origin node with a CTS message. Other nodes, upon hearing this transaction, must remain quiet until an acknowledgement (ACK) message from the receiver is sent, which takes place after the packet data have been sent. Thus the sequence of messages is RTS-CTS-DATA-ACK. The RTS-CTS system is only used for longer frame sizes; smaller packets use only DATA-ACK, with the understanding that some greater probability of collision is balanced against the overhead of the RTS-CTS scheme.

When a hard-wired access point (AP) is available, medium contention can be accomplished via a Point Coordination Function (PCF) instead of DCF. Such a system can provide guarantees on the maximum access delay and minimum transmission bandwidth which autonomous nodes operating under DCF cannot offer. In our applications, this is the kind of service that can be offered when some infrastructure-based communications resources are available, such as with Vehicle-Infrastructure Integration (VII). In a completely mobile network, only autonomous operation is available, through a scheme such as DCF. It is useful to reiterate, however, that much of the concern with contention management is centered on crucial data with specific originators and recipients; anonymous broadcast data of only temporary usefulness may afford much less communications overhead and

simpler protocols. In the applications described in this dissertation, most (if not all) of the data can be described as non-essential data. This is not to say that data are not useful – in fact, as much data as possible makes the system work better. Rather, it only implies that any single piece of data is not so important as to absolutely require its transmission. A failed packet could easily be supplanted by a successful packet sent in a similar traffic environment.

2.1.4 Bluetooth (IEEE 802.15.1)

Bluetooth is a wireless communication scheme designed around the needs of personal devices such as hand-held computers. Bluetooth operates in the ISM (2.4 GHz) band, with frequency-hopping spread spectrum (FHSS), wherein a given transmitter-receiver pair hops around a collection of 79 narrow-band channels in a pseudo-random sequence. The receiver and transmitter follow exactly the same sequence, but it appears random to other nodes, thereby increasing security. The nominal link range in Bluetooth is limited to 10 meters. In theory, because Bluetooth uses a code division multiple access (CDMA) scheme over these channels, a very large number of simultaneous users on each channel is allowed. In practice, however, empirical performance of CDMA has fallen far short of its theoretical capabilities (Murthy and Manoj, 2004) and the Bluetooth community has not figured out how to achieve this performance level either (Tan *et al*., 2001).

It is clear from the essence of the Bluetooth protocol that it is designed around human-initiated and irregular communications. Devices that wish to communicate do so by organizing themselves into "piconets." The first device initiates the process and

becomes the master, while all other nearby devices either go into standby, or enter into the piconet as slaves. Only seven slaves are possible for each master. Communication across piconets is also possible, because a single node can be a master in at most one piconet, but a slave in multiple piconets. This forms what is known as a "scatternet." The common node can only communicate with one piconet at a time because they use different frequency-hopping schemes. Participation in multiple piconets is regulated using a TDMA scheme. Slaves are allowed to communicate only after having been polled by the master. All communications takes place within a time-slotted band with slots of length 0.625 ms.

The limited range, the limited number (thus far) of channels, the limited size of piconets, etc., makes Bluetooth reasonable for small numbers of people making relatively infrequent communications requests. For vehicular applications, particularly with large market penetration, there is a possibility that this combination of limited range and limited frequency division cannot provide enough high-quality data for traffic modeling in real time in congested urban areas.

2.2 Vehicular ad hoc networks

A number of researchers have made specific investigations of the viability of one or more of the above-mentioned wireless ad hoc network protocols to support transportation applications. In this case, since it assumed that (most of) the nodes are located in vehicles, these are called Vehicular Ad hoc Networks (VANETs). Due to the high mobility of vehicles as mobile nodes, the topology of a vehicular ad hoc network can rapidly change and can easily break. The purpose of the

communications might also drive certain considerations. Thus, some additional protocols have been proposed that are specific to vehicular uses. Since this is the likely outcome of this dissertation, these efforts are reviewed here. This section includes a description of the DOLPHIN protocol, followed by a review of some applications in transportation safety and traffic information dissemination.

2.2.1 DOLPHIN

The DOLPHIN (Dedicated Omni–purpose inter–vehicle communication Linkage Protocol for Highway automation) protocol was proposed by Tokuda *et al.* (2000). The main purpose is to support applications such as traffic automation. It is assumed that market penetration is 100%. Each vehicle transmits a set of data relevant to its status on a regular transmission interval. Because many vehicles communicate at once, the data are packetized within that interval, and a CSMA scheme is employed to resolve conflicts. The paper claims to allow for packet collision detection and it does not employ any collision avoidance scheme. The collision detection claim is suspicious, because no details are given, and this is a notoriously difficult problem in wireless communications, as mentioned above. Because the method is only demonstrated using simulation, it is likely that the authors overlooked this fundamental problem. Collided packets are abandoned, which is a potentially useful device for non-mandatory data, which will be explored further in this dissertation.

With the exception of the allowance for, and abandonment of, collided packets, the protocol is just a simplified version of IEEE 802.11. There are a number of suggestions for data content and formatting that are useful in the specific application

of vehicle automation, but these are simply data organization issues that can already be accommodated within the data portion of the IEEE 802.11 packet. The small but interesting conceptual contribution of DOLPHIN is the abandonment of collided packets, but again the lack of consideration of the physics of this problem suggests that the development of the protocol was limited to simulation investigations, and that practical problems would prevent it from being used in reality.

2.2.2 Traffic Safety

Dogan *et al.* (2004) investigated the use of IEEE 802.11 and DOLPHIN protocols for the purpose of intersection collision warning systems. The analysis assumes that all vehicles are equipped with the necessary communications equipment, as well as DGPS (Differential Global Positioning System) hardware and navigation software that allow for precise positioning. The study was conducted on a custom simulation platform.

Vehicle traffic arriving to an intersection is simulated via some simplistic models of driver behavior, stochastic arrival processes, and car-following. It should be noted that the authors' presentation of certain aspects of probability theory is flawed, although the specifics of the arrival distribution are probably not important to the conclusions of the paper. It is unclear how queuing at the traffic light is handled – the paper leaves the impression that each approach is empty, even in the presence of a red light. The particular turning movements that are simulated to generate potential accidents are quite contrived, and this undermines the relevance of the model. The propensity of individuals to get in accidents is an extremely complicated behavioral

issue, one that cannot be handled via such simple models. The appropriate way to test electronic augmentations to the human-machine interface is either via driving simulator (less expensive and somewhat unrealistic) or field test (expensive and dangerous but realistic).

The vehicle paths are used to determine the effect of shadowing on path loss in the wireless signal. Together with other effects, the path loss and fading are simulated using standard models. A very small number of simulations are run. The only performance metric is the rate of packet collision, which suggests that the authors assume that as long as the messages are delivered properly, the intersection collision can be avoided, which is certainly a stretch of the imagination. Furthermore, they conclude that packet losses only occur due to physical layer errors. The problem with this conclusion is that they only modeled a small number of vehicles that might be in the vicinity of the intersection, and they assumed that transmission would only take place within 50 meters of the approach to the intersection. With a realistic number of nearby vehicles (on the road, as well as in parking lots, etc.) with realistic transmission distances and very likely many other purposes for an in-vehicle ad hoc network, one can imagine that the rate of data communications would be orders of magnitude higher than what was simulated in this paper, greatly increasing the likelihood of MAC layer packet collisions. Particularly troubling is the choice to include the DOLPHIN protocol, both because of its obvious limitations mentioned above, as well as the fact that its only real distinction above IEEE 802.11 is its willingness to discard packets, which seems like a very bad idea for safety-oriented

systems. Of course, at the traffic volumes simulated in this paper, the system was probably not taxed to the point that these errors would manifest themselves.

Sawant *et al*. (2004) investigated the use of the Bluetooth protocol for wireless communication on an ad hoc network formed amongst nearby vehicles for the sharing of data from on-board sensors. The authors do not seriously test the limitations of the number of active vehicles, as discussed in Section 2.1.4 of this dissertation. The authors depict accident scenarios at intersections and recognize that while a very specific small set of vehicles should form a common piconet to communicate with each other, the intersection contains many vehicles, none of which know *a priori* how this set should be constructed. Even if this problem were to be solved, the paper assumes that only one such potential conflict can arise, when in fact every vehicle is a possible actor in a wide range of accident scenarios, each of which would conceptually require the formation of a piconet.

The claimed benefit of the system is that by sharing sensed information, vehicles can mutually improve their virtual sensor coverage areas. The problem with this assumption is that all known on-board sensors are very limited in the observations they can make and the conclusions that can be drawn from them. For example, a radar range sensor might be used for obstacle warning and autonomous cruise control on one particular vehicle. This paper argues that predicted object locations, which another vehicle might not be able to sense, could be communicated to that vehicle instead. This is a pleasant thought, but it is fraught with practical problems. For example, the range data are measured relative to the sensing vehicle, and are used for limited purposes. They are usually reasonably accurate in the vehicle-object axis, and

likely very imprecise in an orthogonal direction. This makes the location prediction very unreliable for other vehicles engaged in different and unknown maneuvers with different and unknown trajectories. The authors also assume that precise relative positioning between vehicles can be accomplished via signal strength measurements, which is known to be extremely error-prone.

2.2.3 Traffic Information Dissemination

Ziliaskopoulos and Zhang (2003) propose constructing a distributed traffic information system using an ad hoc network, which they describe as "a zero public infrastructure traffic information system," which simply means that they do not expect to relay on fixed infrastructure as a communications node. The paper investigates various important aspects of such a system, including the speed with which information is disseminated, given different levels of market penetration.

In this paper, however, the underlying modeling is poor. The wireless protocol is claimed to be IEEE 802.11, but in fact no specific details of either the physical or MAC layers are simulated. Communications between vehicles is treated as a deterministic and totally reliable function. Vehicles are assumed only to communicate with vehicles traveling in the opposite direction of a given link, which presumably can be arranged by using directional antennae, although the paper does not specify this. No accounting is made of the possible benefits of sharing data from a wide range of vehicles in the near vicinity, regardless of their current trajectory. The problems posed in this thesis allow for the relaying of data between vehicles without specific knowledge of their trajectories. It would not make sense, in this

case, to rule out vehicles simply because of their direction of travel. On a multi-lane facility, a vehicle will be adjacent to many vehicles traveling in the opposite direction, particularly during congested traffic. This paper assumes that communication between a pair of vehicles must take place within a given window during which they are adjacent to each other, but does not take the other adjacent vehicles into account, nor the fact that they cannot be told apart. This greatly increases the possibility of packet collisions, which severely impacts the performance of the communications system.

Yang (2003) assessed a traffic information system using vehicle–to–vehicle communication based on the Autonet concept proposed by the Institute of Transportation Studies at University of California, Irvine. Vehicles could broadcast information about themselves, the links they traveled on, or incidents, although the author does not specify how a vehicle would ascertain such details about incidents with no human intervention. Of course, no such methods currently exist. Communications is handled very abstractly – each vehicle has a fixed success rate for packets, bandwidth constraints are not modeled, and the probability of success for a packet has nothing to do with the conditions under which it is transmitted; these are all very problematic assumptions. The majority of the dissertation deals with simple exercises in information propagation as a function of market penetration for various idealized roadway geometries. The dissertation focuses on details of the simulation mechanics, but does not address any issues in a more substantive way than papers previously described in this review.

Goel *et al*. (2004) also ask some of the same questions posed in this research. The paper is concerned with using ad hoc networks for traffic information dissemination, and seeks to address the questions of spatial information relevance and required communications bandwidth. The authors used Paramics to simulate a network of uncongested streets on which a single incident has taken place, creating congestion on a single link. Equipped vehicles are able to learn about this congestion from other equipped vehicles, and choose an alternative route. They also ask important questions about when each vehicle should send an information report, and what that report should contain. The choices of what information to disseminate are simple, but very good. In particular, one scheme the authors investigate is to allow vehicles to transmit only "interesting" information, by which they mean information that is markedly different from expected conditions on a link, assuming that vehicles would know such things via their navigation database. This is a very good idea because it limits the amount of useless information clogging up the communications channel.

The authors conclude that bandwidth is not a limiting factor, but their simulation is (incorrectly) constructed to provide this result. First, the paper assumes that a single report of link speed is sufficient to represent the link as a whole, when in fact this is a statistical sample size issue. It would be unwise to make routing decisions based on the (possibly) uncommon experience of a single vehicle. Second, the analysis assumes a fixed rate of dissemination (one broadcast per minute), and does not investigate how this rate should change dynamically to maximize usage of the communications channel with different densities of equipped vehicles. The authors

claim that this is frequent enough because the traffic state does not change with much more resolution than this. While the second part of the argument is true, the conclusion is false, because given the sporadic and transient nature of ad hoc clusters, increasing the rate of dissemination has the effect of reaching a larger sample of vehicles. The clusters could certainly form and disintegrate on a time scale less than a minute. Finally, the authors tested only a single congested link in an otherwise normal network. In most urban areas, during rush hour, all links are congested, and hence all vehicles would likely be transmitting information about all of their link experiences. This raises the information quantity exponentially. If forwarding (relaying) is taken into account (which it should be, given the expected disconnectedness of the network), a further exponential factor can be applied if vehicles are not only transmitting their own experiences, but also relaying those of other vehicles.

Nadeem *et al*. (2004) address the interesting question of information forwarding: how much of another vehicle's experience should a given vehicle broadcast. The authors assume only a small number of vehicles will be communicating with each other, and also make the common mistake of endowing these vehicles with more information than they would have in reality. For example, the paper assumes that a vehicle can conduct its broadcast within a "broadcast period," but the ability of all vehicles to do this depends on the number that desire to communicate, which is known in their simulation but is not known in reality. The bulk of the paper is concerned with algorithms for data aggregation and compression, which are both good ideas but not the concern of this research. It should be pointed out that data

aggregation is problematic without specific understanding of the applications to which the data will be applied by receiving vehicles. In this paper, the authors choose aggregation schemes according to the communications constraint, without consideration of the fact that excessive or improper aggregation will result in useless data.

Wu *et al*. (2005) and Wu (2005) test information dissemination on the I-75 corridor in Atlanta, as simulated using Corsim's Real Time Extension (RTE), supplemented with the communications simulator QualNet. In their simulation testbed, they assumed two unrealistic environments. First, they did not allow equipped vehicles to dynamically reroute based on disseminated traffic information (presumably the purpose of such a system); equipped vehicles learn about congestion from other equipped vehicles, and do not choose an alternative route. Second, they did not consider the MAC layer, which is a very important layer. While they mentioned that their testbed is "neutralized" on the MAC layer, communication collision is not simulated on their testbed. They measure the rate of information propagation for a single message across a network, assuming perfect conditions for relaying (forwarding) between vehicle clusters, without accounting for the fact that message traffic will limit this capability. Thus, while the methods they employ are generally acceptable, the question posed is not a very meaningful one.

Wischhof *et al.* (2005) proposed methods for scalable information dissemination in mobile ad hoc networks. They employed the network simulator ns-2, augmented with a vehicle movement model based on cellular automata. The authors considered an important MAC level change similar to what is proposed in this research: all data

packets are transmitted in the form of local (single hop) broadcasts. Nodes are never directly addressed and no routing of data packets in the traditional sense is performed. Again, the research in this case is not concerned with the value of the information for various applications, or the rate at which that value diminishes in time and space.

Xu and Barth (2006) proposed travel time estimation techniques for traffic information systems based on intervehicle communications. As a travel time estimation technique, they used a decay factor to weight the "freshness" of data and experimented with the model on a simulation testbed with Paramics and ns-2. In this paper, they defined a road segment as a stretch of a road between two successive exit points such as junctions or exits. Their road network model, in which an interchange is represented by a single node, is too simple to describe realistic congestion situations. Considering that congestion typically starts from a merging or split area, road segments and ramps in interchanges should have been dealt with independently.

Saito *et al*. (2007) proposed an intervehicle information dissemination protocol called Received Message-Dependent Protocol (RMDP) which autonomously changes the broadcast interval in order to avoid the "broadcast storm" problem that might occur when vehicles cannot develop a sense of the amount of competing communications traffic. The broadcast interval changes depending on the number of received messages and reception errors. They evaluated their protocol on a simulation framework in which a transportation simulator, NETSTREEAM, and a communication simulator, MobiREAL, are combined. In their experiment, a heavy traffic condition and a light traffic condition were simulated at an intersection and on

an urban road network, respectively. They did not, however, allow vehicles to dynamically reroute based on disseminated traffic information.

2.2.4 Vehicle-Infrastructure Integration

Most of the reviews related to VANETs have focused on experiments for vehicle–to–vehicle communication alone. Hybrid systems are also possible, which might incorporate fixed infrastructure to serve one of several possible functions – as an access point to connect the ad hoc network to a wired network, as a congestion mediation device for wireless traffic, and as a consolidation / aggregation point for traffic data. The fixed nodes could also play the role of patching together otherwise disconnected vehicle clusters, although this would obviously happen randomly. The U.S. Department of Transportation announced new major initiatives to aim at improving transportation safety, relieving congestion and enhancing productivity at the 2004 ITS America Annual Meeting (US DOT, 2007). Vehicle Infrastructure Integration (VII), one of these major initiatives, aims to achieve nationwide deployment of a communications infrastructure on the roadways and in all production vehicles through vehicle–to–vehicle communication and vehicle–to– infrastructure communication (US DOT, 2007). Figure 2-1 shows the VII Architecture proposed by US DOT.

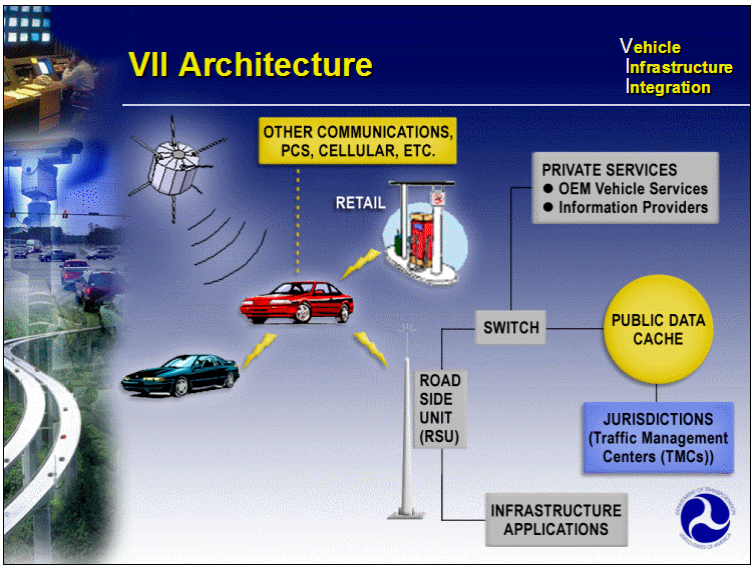


FIGURE 2-1 US DOT's VII Architecture (Werner, 2004)

The primary thrust of the VII to date has been a vehicle → infrastructure → vehicle paradigm. Individual vehicles would serve as probes, reporting their findings back to roadside units (RSU) at opportune times. With enough such data, centralized applications could generate estimates and perform other applications. Data would then be transmitted back to vehicles via the RSUs.

The establishment of the VII Architecture by US DOT builds on other research and operational tests. Of course, vehicle manufacturers would install the technology in all new vehicles, necessitating some standards. The manufacturers have conducted a number of experiments to help flesh out what the system parameters should be. The group involved in discussions on the VII Initiative comprises the VII Coalition, a cooperative effort between public and private sectors:

- US DOT - FHWA, FMCSA, ITS JPO and NHTSA

- Automotive Manufacturers - BMW, Daimler Chrysler, Ford, General Motors, Honda, Nissan, Toyota Motor North America, and Volkswagen
- State/Local Agencies - CALTRANS, Florida DOT, Idaho DOT, Indiana DOT, Maryland State Highway Administration, Metropolitan Transportation Commission (San Francisco Bay Area), Michigan DOT, Minnesota DOT, New York State DOT, Utah DOT, Virginia DOT, and Washington State DOT
- Associations - AASHTO, Alliance of Automobile Manufacturers, Association of International Automobile Manufacturers, IBTTA, ITE, and ITS America (ITS America , 2005).

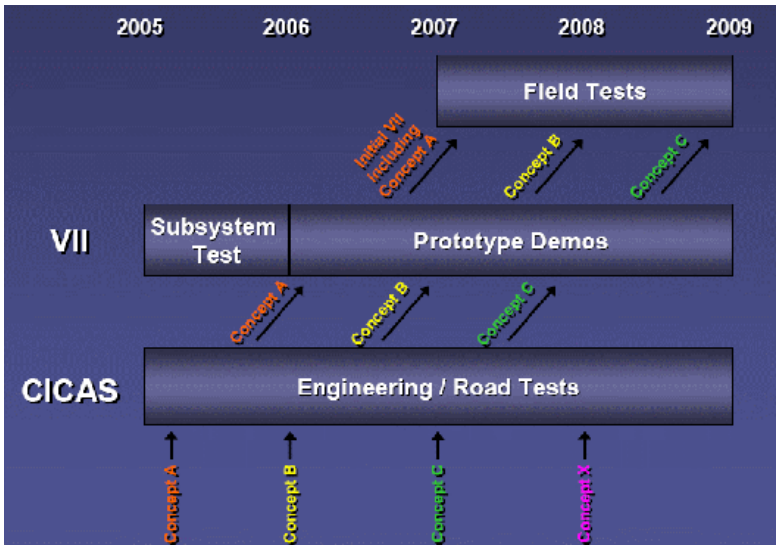


FIGURE 2-2 VII timeline

A VII Coalition has been established to determine the feasibility of widespread deployment and to establish an implementation strategy. As shown in Figure 2-2, a general timeline has been developed, and balanced works between public and private sectors have been conducted under the timeline (Werner, 2005).

2.3 Traffic information imputation

For any given traffic model, such as a link speed estimation model or an optimal route choice model, one could postulate the set and characteristics of a data stream that would provide the greatest performance. Of course, different models have different levels of sensitivity to changes in these characteristics. Assuming that all data come from the ad hoc mobile network (and, possibly, some fixed infrastructure stations), it is clear that there will be limitations on the quantity and quality of data that can be delivered. One part of this dissertation will be dedicated to determining what those data limitations might be, as a function of market penetration and communications systems constraints. In some cases, it is expected that recommendations could be made as to protocol design, that would minimize the deleterious impacts of data shortcomings. In any event, it will be possible to assess the performance degradation of the application as a function of the data degradation.

To counter this effect, it is proposed that "missing" data be imputed from surrounding data in time and space. For any particular model, and given the physics of traffic dynamics, the usefulness of data is expected to decline in both time and space, although this effect has not been studied systematically. A few very specific proposals for data imputation have been developed, and they are reviewed in subsequent paragraphs. It should be noted, however, that none of these efforts was conducted with the goal of guiding the development of an appropriate wireless data provision mechanism; in fact, most assume that data are provided by fixed detectors. Furthermore, in most cases, the methods are proposed in order to maximize the number of data points that can be extracted and made useful from ITS data archives.

The subject of information relevance in quasi-real-time applications is therefore relatively untouched, and the results from these papers might only be tangentially useful to the proposed research effort.

Smith *et al*. (2003) introduced three types of heuristic techniques for imputing missing speed, occupancy, and flow data, collected from loop detectors or similar hardware. The techniques include historical averaging, spatial averaging, and temporal averaging. The historical average technique substitutes missing data from historical averages over previous days, weeks, months, etc. Clearly, this method can misrepresent conditions in congestion or accidents. Also, it is not feasible in a pure ad hoc network setting, since individual vehicles would not possess network-wide historical data. This is one function that fixed infrastructure nodes might serve.

The spatial averaging method attempted involved the weighted average of surrounding detectors with historically based lane distributions. The data from nearby links would be available to wireless nodes, but again the historical information would likely not be available. Temporal averaging over recent data is more appropriate in the wireless setting, because it is possible for a wireless node to have the necessary data. To make full use of this method, however, would require local storage of recent data on the appropriate set of links. Short-term changes are not necessarily fluctuations; they could represent systemic changes in traffic state due to the passage of shock waves, for example. Such a method would have to realize that older data might be biased, and some effort should be made to correct for this. The proposed research on temporal information degradation would be useful to determine

the length of time that specific data should be retained, as well as patterns of bias that might be expected and countered for.

One of the most troublesome traffic sensors is the inductive loop detector, because it requires frequent tuning to make sure that inductance thresholds correspond properly to vehicle passage. Most highway agencies are not able to keep up with the maintenance requirements, and as a result, there are many loop detectors delivering inaccurate data. Alarmingly, the loop detector is also the most common sensor. Chen *et al*. (2003) and Al-Deek and Chandra (2004) investigated the situation where data are missing from the middle of a sequence of three detectors. They used pair–wise regression models to impute missing data from dual–loop detectors, which assumes a linear (in parameters) and statistical relation between the measurements at the detectors. In fact, detector measurement differences in closed systems results entirely from shockwave propagation and differing density and flow conditions along the link. Newell's kinematic wave theory (Newell, 1993) addresses this problem exactly, except for minor statistical fluctuations that might result from counter errors or lane changes. This latter method, which exploits the physics of traffic dynamics, is much more explanatory than a statistical model that captures correlations that occur by happenstance.

Gold *et al*. (2001) explored a variety of the above methods. They describe something they call "factor up," which in fact is temporal averaging over a fixed time window of moderate length, and "interpolation," by which they mean averaging over two temporally adjacent observations, which is clearly a variable-length window. They also use regression methods. Importantly, they acknowledge the bias induced

by using old data in the presence of systemic traffic state changes, as described previously in this proposal, and they suggest that some weighted averaging scheme could be used to correct for this (which is true), but do not pursue the idea any further.

2.4 Discussion

It is clear from the above review that the general topic of ad hoc wireless networks for vehicular purposes is of great contemporary interest. Due to the complexities of traffic and communications, simulation is the most common, and most appropriate, analysis tool. A few papers offer analytical solutions for grossly simplified problems that are simply not instructive.

The subject is nowhere close to mature, and there are many ripe opportunities for important research. The primary goal behind this research is to make strides in an integrated transportation and communication simulation framework development and performance assessment that recognize the important nature of traffic-related information that is broadcast anonymously. The most important findings from the literature review are as follows:

- Vehicular ad hoc networks dealt with in this study is a novel and promising approach to transcend the limitations of traditional transportation systems although there is no case applied to a real transportation system.
- Many studies related to VANETs have been conducted. Although computer simulation is a popular evaluation method in those studies, it still confronts a major challenge in terms of “reliability,” the degree to which it replicates a

real system. For reliable demonstration, realistic assumptions for transportation and communication are required.

- Even though several simulation frameworks for a VANET-based traffic information system were developed in previous studies, no one showed practical experiments and evaluation results. In particular, vehicles' rerouting based on shared information would be a key output fed back to the transportation system.

In a traffic information system on which this research focuses, vehicles would collect, share, and feed back traffic information. Developing a simulation framework, it is important to define the characteristics of traffic information since a framework design could be changed according to the definition. The next chapter discusses about the characteristics of traffic information. Individual data and traffic conditions corresponding to each single data are described with individual travel times from a simulation experiment, and then, temporal and spatial relevance on aggregate travel times from another simulation experiment are observed.

Chapter 3: Traffic Information Characteristics

In a traffic information system based on a VANET, traffic information would be collected, shared, and used as it cycles. When a simulation framework for such a system is designed, traffic information such as travel time, speed, vehicle location, etc. could be accumulated in database, estimated to impute missing data, aggregated to obtain a representative of a certain situation, and removed if it is too stale to use. It is important to understand the characteristics of traffic information since traffic information is processed for various purposes in a simulation framework. This chapter discusses preliminary researches about the reliability and relevance degradation of travel time information as its characteristics. Section 3.1 explains the relation between individual travel time data and aggregated data, and it describes the quantity for reliable travel time information. Section 3.2 shows temporal and spatial relevance degradation among travel time data. All travel time data used in this chapter were obtained from simulation experiments using only a transportation simulator, and a communication simulator would be dealt in the simulation framework.

3.1 Individual travel time characteristics

In a VANET-based traffic information system, individual travel time data may be dealt with, compared to average data (usually 1 minute or 5 minute aggregation intervals) used in traditional traffic information systems. Taking into account a low market penetration, the data obtained from vehicular ad hoc networks could be too sparse to apply as representative of traffic conditions on a certain link. The

conclusion is that sampling errors for travel time information and differences between individual data and an average of data would exist as implicit weak points, and would influence the reliability of individual travel time information. In order to investigate the reliability of individual travel time data, this section shows patterns of individual travel time in Subsection 3.1.2 (Kim *et al*., 2007a) and data quantity for reliable travel time in Subsection 3.1.3 (Kim *et al*., 2007b) based on the results obtained from a simulation experiment in subsection 3.1.1.

3.1.1 Experiment for individual travel time

It should be noted that because a simulation environment is used, one can assume that the data collection process is comprehensive and accurate, which of course is not true in reality. It is used, therefore, as a "ground truth" of sorts, recognizing the standard pitfall of simulation models, which is that it represents only the truth of how the simulation logic attempts to produce realistic driver and vehicle behavior, rather than the truth associated with real cars and drivers. Since real data collection mechanisms of this sort do not yet exist, of course simulation is the only way to produce these data.

If we measure individual travel times of all vehicles passing a certain link using, for example, a license plate matching technique, we can obtain individual travel time data and calculate their aggregates. Figure 3-1 illustrates how to pair up individual travel time data and a vector of aggregate information, assuming we obtain individual travel times of all vehicles on a certain link.

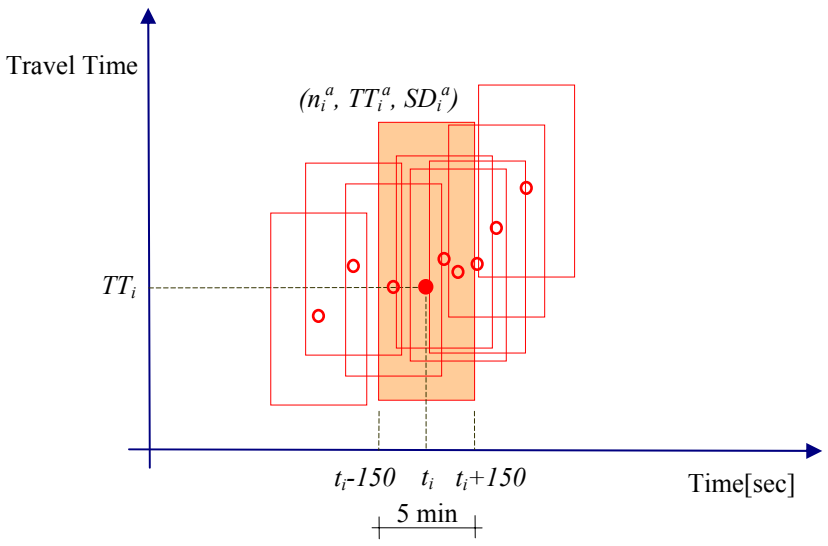


FIGURE 3-1 Illustration of a datum set for vehicle *i*

As Figure 3-1 shows, every datum is represented as an ordered triplet of an individual travel time for a single vehicle, a time stamp, and a vector of aggregate information for the link and time window occupied by that vehicle at the time the datum was collected. Aggregate information is used as a proxy for actual information representative of a traffic condition: the quantity of data, and the mean and standard deviation of all individual data within a 5-minute window centered on the time instant in question.

$$d_i = \left[TT_i, t_i, \left(n_i^a, TT_i^a, SD_i^a \right) \right] \quad (1)$$

where

d_i: datum set for vehicle *i*,

TT_i: individual travel time for vehicle *i* on a certain link *l*,

t_i: time stamp (arrival time) for vehicle *i*,

n_i^a : number of data in the aggregation window for vehicle i,

TT_i^a : travel time mean in the aggregation window for vehicle i on link l, and

SD_i^a : variance of travel time in the aggregation window for vehicle i.

In order to set up datum sets of Equation (1), it is necessary to obtain an "entire" set of space-based travel time data for a specific time period. A simulation experiment was conducted on a real road network for two hours. Paramics 5.2 (Paramics homepage, 2007), a microscopic traffic simulator, was employed. Through its API (Application Programming Interface), entry times and exit times of all vehicles which arrive at and leave the target link were recorded respectively. Individual travel time data were extracted from the difference between the entry time and the exit time and were matched up with aggregated travel time data. Figure 3-2 shows the experiment site.

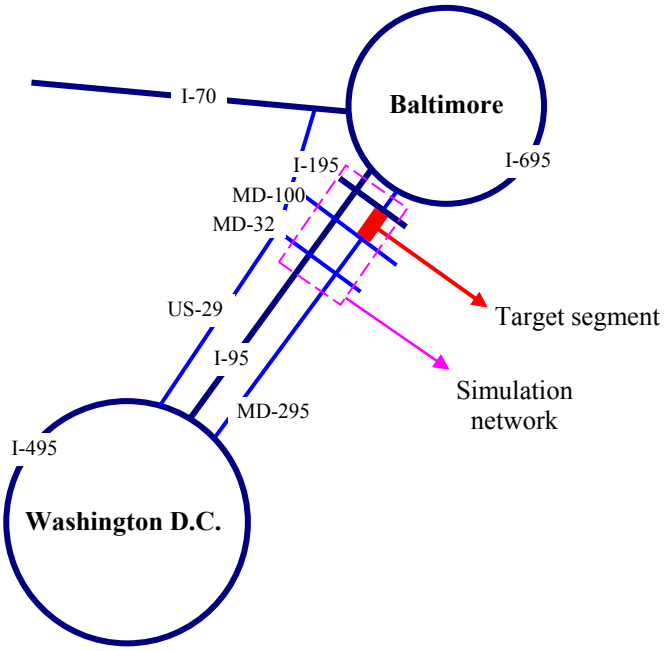


FIGURE 3-2 Experiment site

In Figure 3-2, the site selected is located on the northbound direction of the Baltimore-Washington Parkway (MD-295) in the state of Maryland, U.S. Details on traffic demands and building the road network model were mentioned in Section 6.1. The northbound target segment (2.24 miles) on the road network was chosen, and individual travel times from all vehicles passing that segment were measured. Table 3-1 shows general observations measured.

TABLE 3-1 General observations for entire individual data set

	Individual travel time	Average		
		Travel time	# of data	Standard Deviation
Number of data	4,443	4,299	4,299	4,299
Maximum [second]	996	886	302	102
Minimum [second]	111	140	18	9
Median [second]	194	208	230	27
Mean [second]	280	264	218	33
Standard deviation [second]	209	182	46	21

As shown in Table 3-1, individual travel time data were measured from 4,443 vehicles passing the target segment. Of those, only 4,299 data points were far enough from the simulation begin or end times that they could be situated inside time windows for which averages could be computed; 144 data points (27 at the beginning of the simulation and 117 at the end) were excluded to avoid end effects due to aggregation in the finite time window. Standard deviations were used as a measure of dispersion instead of variance to be consistent with the units of travel time.

Based on the datum sets consisting of individual data and its aggregates obtained from the simulation experiment, patterns of individual travel time were explored in Subsection 3.1.2, and data quantity for reliable travel time information was dealt with in Subsection 3.1.3.

3.1.2 Reliability of individual travel time information

In order to observe the representative degree of individual travel time data for traffic conditions, this study paired up individual travel times and 5-minute aggregates of travel time. Data obtained from the simulation experiment in Subsection 3.1.1 were used.

Individual travel time data were provided from each vehicle, mimicking what would have been obtained from an vehicular ad hoc network in place. Surrounding each of these travel time reports from individual vehicles, the average data calculated from individual travel times within a 5-minute window containing that single data point were observed. Figure 3-3 shows the pattern of the entire individual travel time data obtained from the experiment in Subsection 3.1.1.

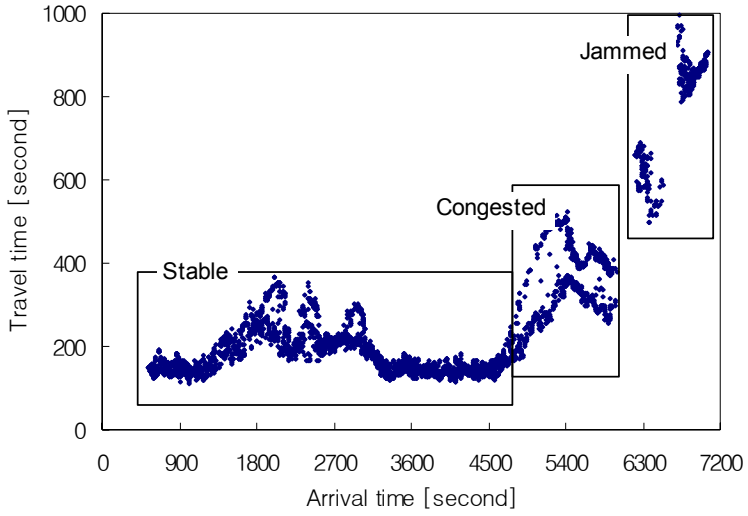


FIGURE 3-3 Entire individual travel time data set

As shown in Figure 3-3, traffic flow was stable until 4,800 seconds (1 hour 20 minutes) after the simulation started, although a small delay happened around 2,100 seconds due to the temporary presence of a queue on a ramp. During seconds 4,900 to 6,000, travel time increased abruptly and the plotted data were visually disconnected. At this point, vehicles were totally in the middle of a jam though they moved intermittently around 6,300 and 6,900 seconds.

As described previously, aggregate travel time data within 5-minute windows were used to represent the ground truth values of travel time. Figures 3-4, 3-5 and 3-6 show the quantity of data, travel time means, and standard deviations as aggregate information of individual data according to simulation time, respectively. In those figures, all data were classified into three groups according to traffic conditions; those of Groups 1, 2, and 3 are stable, congested and jammed, respectively.

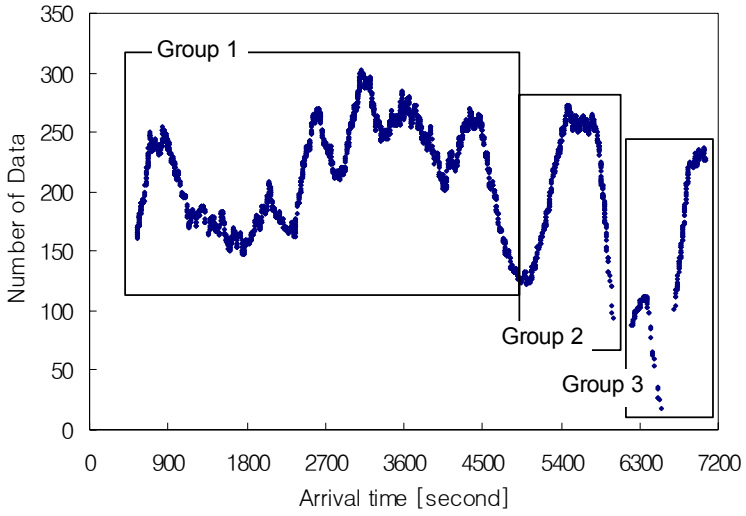


FIGURE 3-4 Quantity of individual data

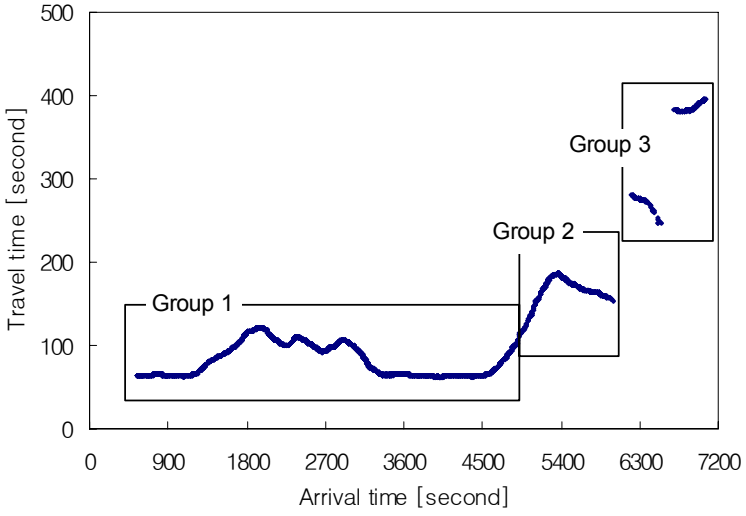


FIGURE 3-5 Travel time means

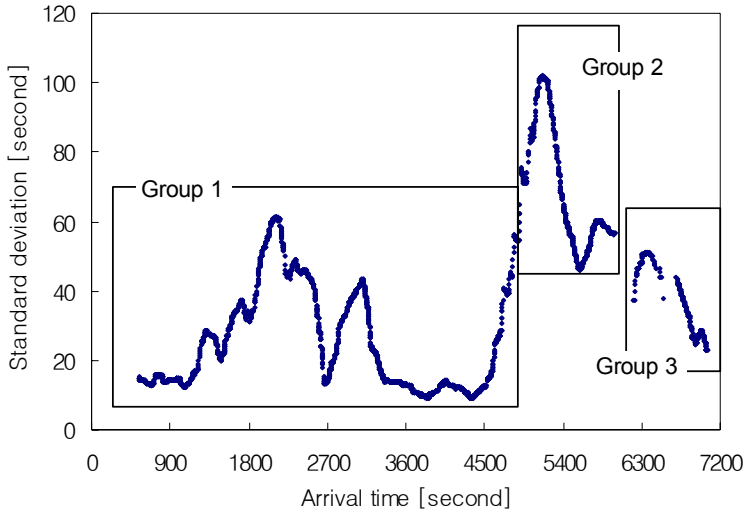


FIGURE 3-6 Standard deviations of travel time

As shown in Figure 3-4, the quantity of data abruptly increased during 5,009 seconds (122 data) to 5,475 seconds (272 data) and decreased from 5,756 seconds (265 data) to 5,995 seconds (94 data). The description of the traffic situation experienced by Group 2 is that traffic flow reached the maximum flow rate, congestion started, and traffic, finally, jammed up. Figure 3-5 also shows a pattern similar to Group 2 in Figure 3-4. That pattern is clearer in Figure 3-6. In Group 2, the standard deviation steeply rose and fell before reaching road capacity. Group 3 is in a severe congestion condition: high travel time but low standard deviation. Figures 3-7 and 3-8 contain the number of data and standard deviations corresponding to each travel time mean respectively.

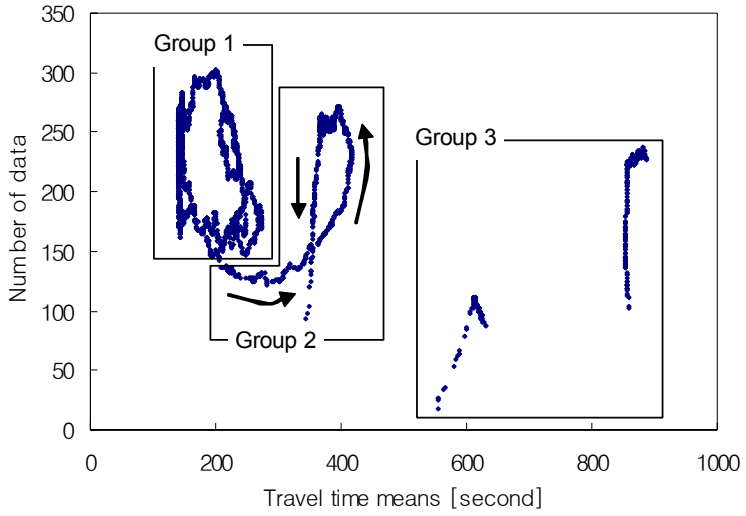


FIGURE 3-7 Travel time means vs. number of data

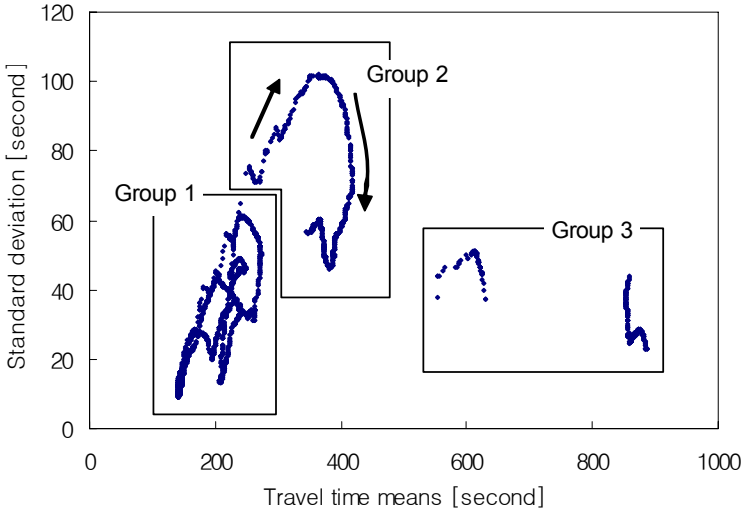


FIGURE 3-8 Travel time means vs. standard deviation

In Figures 3-7 and 3-8, travel time data are clearly classified even though those are not in time order; these three groups are distributed according to traffic conditions. Data in Group 1 occupies a small area with the largest number of

samples. On the other hand, the data in Group 2 are on a curve and they follow the arrows, chronologically.

Less so than aggregate data, individual data would be widely spread because they are statistically distributed with error. Figure 3-9 plots individual travel time and correspondent travel time means, and Figure 3-10 shows the relation between individual travel time and standard deviation.

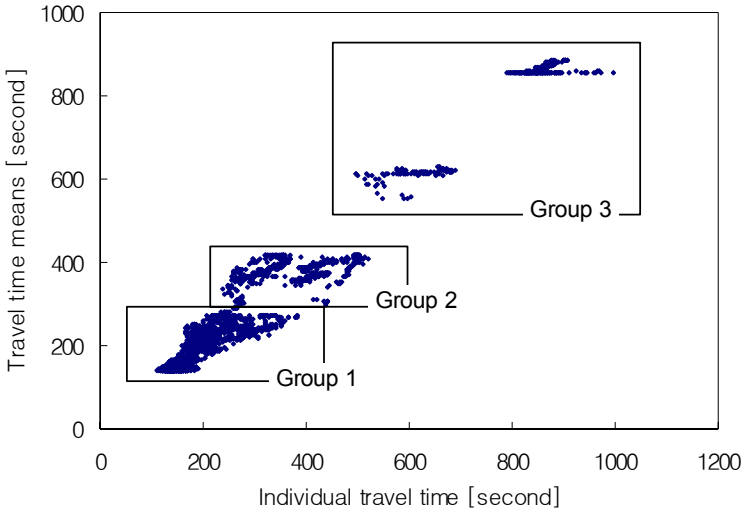


FIGURE 3-9 Individual vs. travel time means

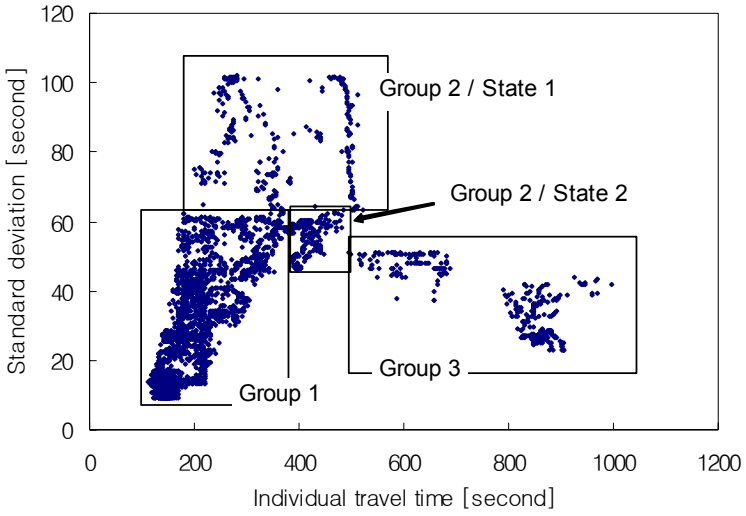


FIGURE 3-10 Individual data vs. standard deviations

As expected, Figure 3-9 shows large variance in Group 2 that would reduce the accuracy of individual travel time information provided in a vehicular ad hoc network. In order to reduce the range of the less reliable information, this study broke up Group 2 into a small-variance group and a large-variance group as shown in Figure 3-10. Finally, a total of four clusters for individual travel time data, Group 1, States 1 and 2 in Group 2, and Group 3, were defined. Table 3-2 summarizes observations of the four clusters based on individual travel times and corresponding standard deviations.

TABLE 3-2 Traffic condition states

	Traffic flow characteristics	Standard Deviation	Experiment Result	
			Time stamp [sec]	Travel time [sec]
Group 1	Uncongested traffic condition before reaching maximum flow. Traffic flow is stable.	Small	546 - 4,900	111-373
State 1 / Group 2	Transition period to congestion. It happens when congestion begins. Travel time changes very quickly.	Large	4,901 - 5,427	201-522
State 2 / Group 2	Congested traffic condition. Vehicles stop and go repeatedly.	Small	5,429 - 5,995	380-481
Group 3	Severe congestion condition. Vehicles are in jam for a majority of time. Standard deviations are, however, low because entire vehicles linger.	Small	6,213 - 7,049	499-996

As Table 3-2 shows, for any standard deviations except State 1 in Group 2 it is reasonable to use individual travel times (collected via the ad hoc network) as a surrogate for average travel time, the actual value that drivers presumably want to know. Individual data included in State 1 in Group 2 occupied a large range of travel time (201 – 522 second) in spite of the small number of data (293 in this study). These data occurred only when a traffic condition grew steeply worse (4,901 – 5,427 second). The range of State 1 in Group 2 accounted for a wide time period because this study dealt with only a single travel time sample. Various approaches with more samples or different information sources, for example from infrastructure, would be expected to be able to minimize unreliable data.

Given the simulation used is a reasonable proxy for reality, the results highlight the simple but important data quality issues that should be considered in the simulation framework for a VANET-based traffic information system. The next subsection introduces data quantity for reliable travel time information.

3.1.3 Acceptance probability

The key question in this subsection is how many individual travel time data we need in order to obtain reliable traffic information for a certain road link in VANETs. Some studies related to data quantity for traffic information have been conducted using various statistical sampling methodologies (Srinivasan and Jovanis, 1996 and Wang et al., 2005). These papers exploit Central Limit Theorem arguments to develop minimum sample sizes according to the familiar inequality:

$$n \geq \left(Z_{\alpha/2} \cdot \frac{\sigma}{d} \right)^2 \qquad (2)$$

where n is the minimum sample size, Z is the standard normal distribution, α is significance level, σ is the standard deviation of the population, and d is maximum allowable error difference.

The statistical precondition for Equation (2) is that population is normally distributed (not likely in most traffic measurements) or that the sample size is large enough to benefit from the Central Limit Theorem. By these reasons, Chen and Chien (2000) obtained the minimum number of probe vehicles through heuristic

methods as well as statistical ones. Of course, individual travel time data are unlikely to be normally distributed, although some experiments have found this to be a good distributional fit in uncongested conditions. In congested conditions when this is not true, rapidly changing conditions affect sample sizes in a way that might invalidate the Central Limit Theorem. In such cases, we propose the Acceptance Probability method as a distribution-free alternative.

The fundamental principle of Acceptance Probability is the same as Equation (2) with the exception of the size of the sample. This describes the probability that an individual datum is within an allowable error range of the median for all individual data within a certain time window. What we want to know is how well a single individual datum represents a traffic condition corresponding to the datum chosen. In this study, we use the median of all individual data within a time window centered on the time instant in question as a proxy for actual information (ground truth). The formulation of Acceptance Probability is defined by Equation (3) as:

$$\mathrm{P}\left(\left| TT_i - median_t \right| > MAER \cdot median_t \right) \tag{3}$$

where TT_i is the individual travel time for vehicle i, median t is taken over travel time data within a certain time window corresponding to time t, and Maximum Allowable Error Rate (MAER) is the maximum allowable error rate.

Data obtained from the simulation experiment in Subsection 3.1.1 were used to evaluate the Acceptance Probability. Individual data collected for a short time period (5 minutes in this study) were aggregated as a proxy for actual information

representing a traffic condition: the median of travel times (TT_i) of all individual data within a 5-minute window centered on the time t. All individual data were matched up with their corresponding aggregate information. For more continuous changes of traffic conditions, medians corresponding to each datum are extracted.

Minimum sample sizes for estimating a population mean were obtained through Equation (2), with significance level, α, set to 0.05 and maximum allowable error difference, d, was MAER*median. MAER used in both Equations (2) and (3) was chosen to be 15 %. Those are for data within 5-minute time windows centered on each individual travel time datum obtained from simulation. Figure 3-11 shows minimum sample sizes corresponding to each time stamp.

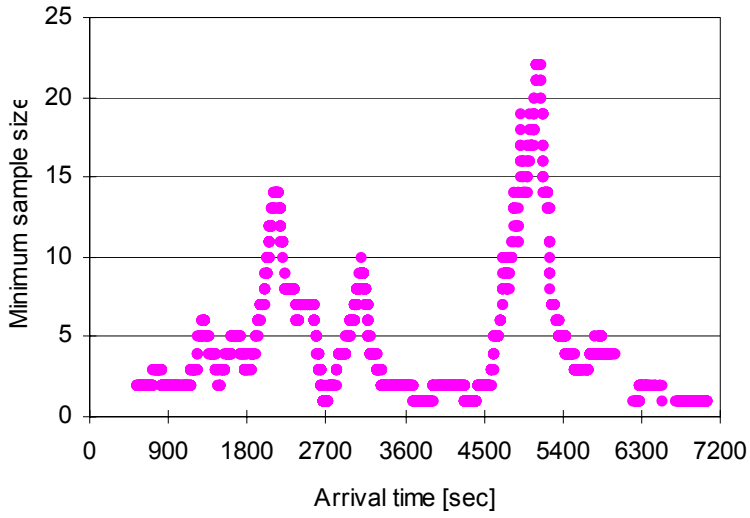


FIGURE 3-11 Statistical minimum sample size

As mentioned above, Equation (2) relies on the Central Limit Theorem, which cannot always be supported. As shown in Figure 3-11, a sample of size 2 can

ostensibly represent population means in stable traffic conditions from 600 to 1,200 seconds and from 3,300 to 4,500 seconds, during which time vehicles drove close to the speed limit (55 mph). Even in jammed conditions after 6,300 seconds, similar results were obtained due to small variance. In the case that many samples were required, e.g. 22 samples around 5,100 seconds, as well, the Central Limit Theorem is well supported.

In order to estimate data quantity for reliable travel time information regardless of the data distribution in VANET, Acceptance Probability from Equation (3) was applied. The probability values obtained from Acceptance Probability imply how well a single travel time datum represents a traffic condition at the time corresponding to that datum. Further, high probabilities would be associated with small sample size and low probabilities correspond to large sample size. Figure 3-12 shows Acceptance Probability for individual travel time data.

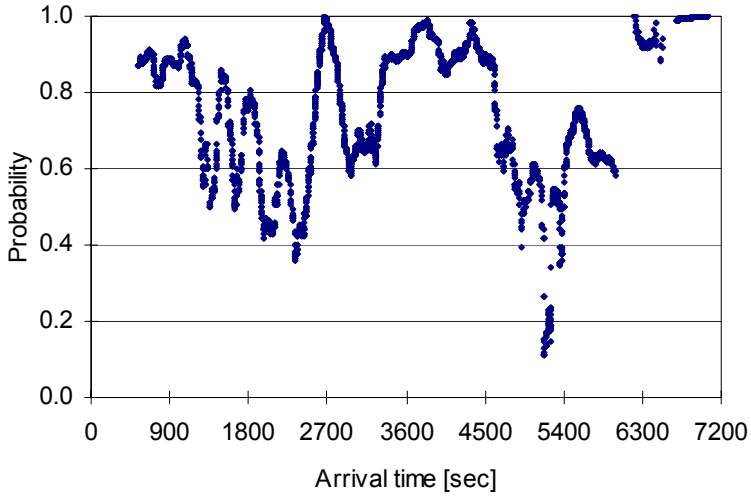


FIGURE 3-12 Acceptance probability

In Figure 3-12, the results for Acceptance Probability show that it is analogous to the statistical method in both stable traffic conditions (600 – 1,200 seconds and 3,300 – 4,500 seconds) and jammed conditions (above 6,300 seconds), when the probabilities are over 0.8. The data around 5,100 seconds seem to be less reliable (under 0.2 probability), which relates to the large sample size required of the statistical method.

The two approaches can be compared more directly by superimposing the minimum sample size results from Figure 3-11 with the complements of the probabilities from Figure 3-12, since we expect that high probabilities would be associated with small sample sizes and low probabilities with large sample sizes. Figure 3-13 shows these minimum sample sizes and (complementary) Acceptance Probabilities for individual travel time data within a 5-minute time window centered on each datum.

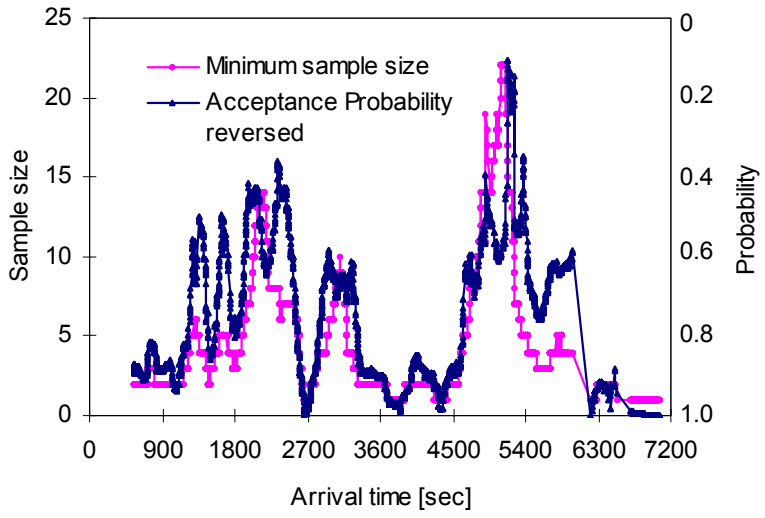


FIGURE 3-13 Statistical minimum sample size and reversed Acceptance Probability

In Figure 3-13, these two profiles match quite closely during uncongested conditions, when the underlying data might be expected to be normally distributed, or during consistently congested conditions, when the same steady state persists long enough to generate large sample sizes, so the Central Limit Theorem can be invoked. During transient (unstable) congested periods, however, the results can differ significantly, and the safest conclusion to draw here is that minimum sample sizes should be increased beyond what the standard methods would suggest. The largest sample size in this experiment was 23 in congested traffic condition.

This study showed an experimental way to obtain the minimum quantity of data required for reliable travel time measurement over a variety of traffic conditions, adjusting minimum sample sizes obtained from a traditional statistical equation through the Acceptance Probability distribution-free method. This idea is expected to help determine data quantity for reliable travel time information on a congested traffic condition in a simulation framework design.

3.2 Travel time information relevance

This section discusses spatial and temporal relevance on travel time information. Travel time data may be located on space and time. Data from nearby links might be useful, but perhaps less useful as the links grow more distant. Similarly, recent data are useful, but they become stale in time. This idea relies on the assumption that traffic data are strongly correlated with each other in terms of space and time. This is true in various degrees for different network topologies. Figure 3-14 shows a conceptual representation of this idea.

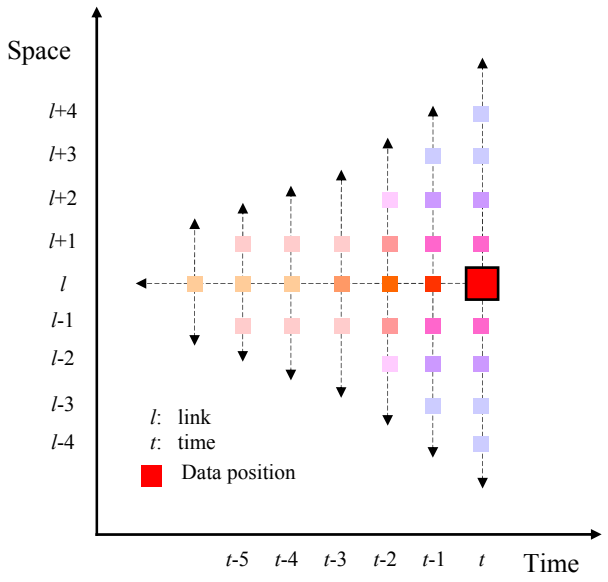


FIGURE 3-14 Concept for spatial and temporal relevance of data

In general, spatially neighboring data are from upstream and downstream links, and temporally neighboring data are from recent historic time windows. We expect these correlations to decline as space-time cells become more "distant," in either the dimensions alone or combined. In order to demarcate this mesh of points, the space dimension will be carved into physically convenient "links." The time dimension, on the other hand, is not physically constrained, but an important decision has to be made. In order to clarify this concept, a simulation experiment for obtaining travel time data is described in the next subsection. Using travel time data obtained from this experiment, spatial and temporal relevance of travel time data is examined (Kim and Lovell, 2006a) and an experimental linear model is introduced (Kim and Lovell, 2006b).

3.2.1 Experiment for spatial and temporal relevance

This subsection explains a simulation experiment to investigate the spatial and temporal relevance of travel time information. The experiment was conducted in a simulation environment in which the transportation simulator, Paramics, was used same as Section 3.1. Figure 3-15 shows the virtual simulation road network used for this study.

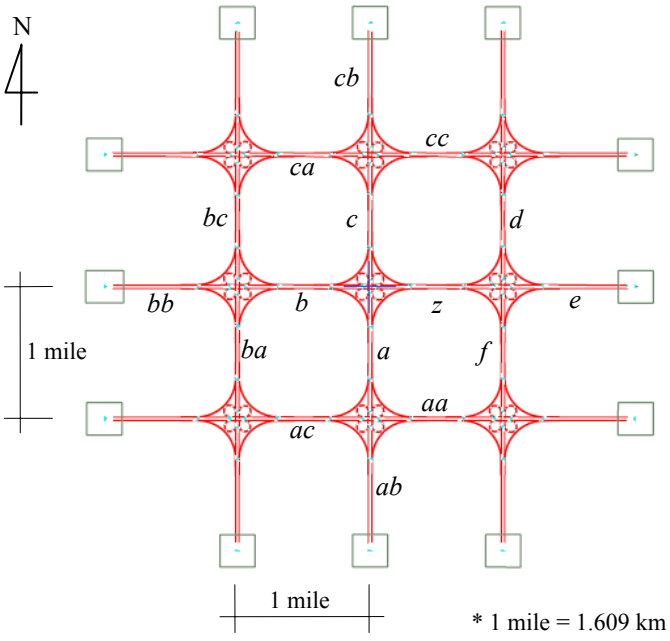


FIGURE 3-15 Simulation network

In Figure 3-15, the simulation network consists of an 8-lane uninterrupted highway (4 lanes per direction) with 12 zones which generate demands. All interchanges are complete cloverleafs without signals. In order to observe spatial and temporal relevance, we chose a target link and neighboring links whose travel times were obtained in this study, as indicated in Figure 3-15. The target link *z* is eastbound

and is located in the center area of the network. It is connected to links *a*, *b* and *c* (upstream) and to links *d*, *e* and *f* (downstream). These are all links that are topologically adjacent to link *z*. Additional links that are one step further removed from the target link are named *aa*, *ab*, and *ac* before link *a*; *ba*, *bb*, and *bc* before link *b*; and *ca*, *cb*, and *cc* before link *c*.

The simulation experiment was conducted for two hours. In order to study a worst case, we would like to observe vehicles moving slowly (maximum volume). However, this is an unstable state that can change rapidly. It was decided that every zone generates 4,800 vehicles per hour. In this study, one-minute space mean speeds, the reciprocal of travel time, on links are used as traffic data to observe spatial and temporal relevance.

It was very important to introduce changes in traffic that prevent accidental correlation. For example, one could impose a very mild steady state condition on the network, in which case adjacent links would have strongly correlated travel times simply because free-flow travel times would dominate. Instead, it was important to introduce non-stationary changes of sufficient magnitude across a variety of origin-destination pairs to minimize this risk; thus, correlations that appear in the data are structural and therefore important. From the results of the simulation experiment for two hours, 120 one-minute space mean speeds by link were extracted. Figure 3-16 shows traffic variations for a target link and neighboring links.

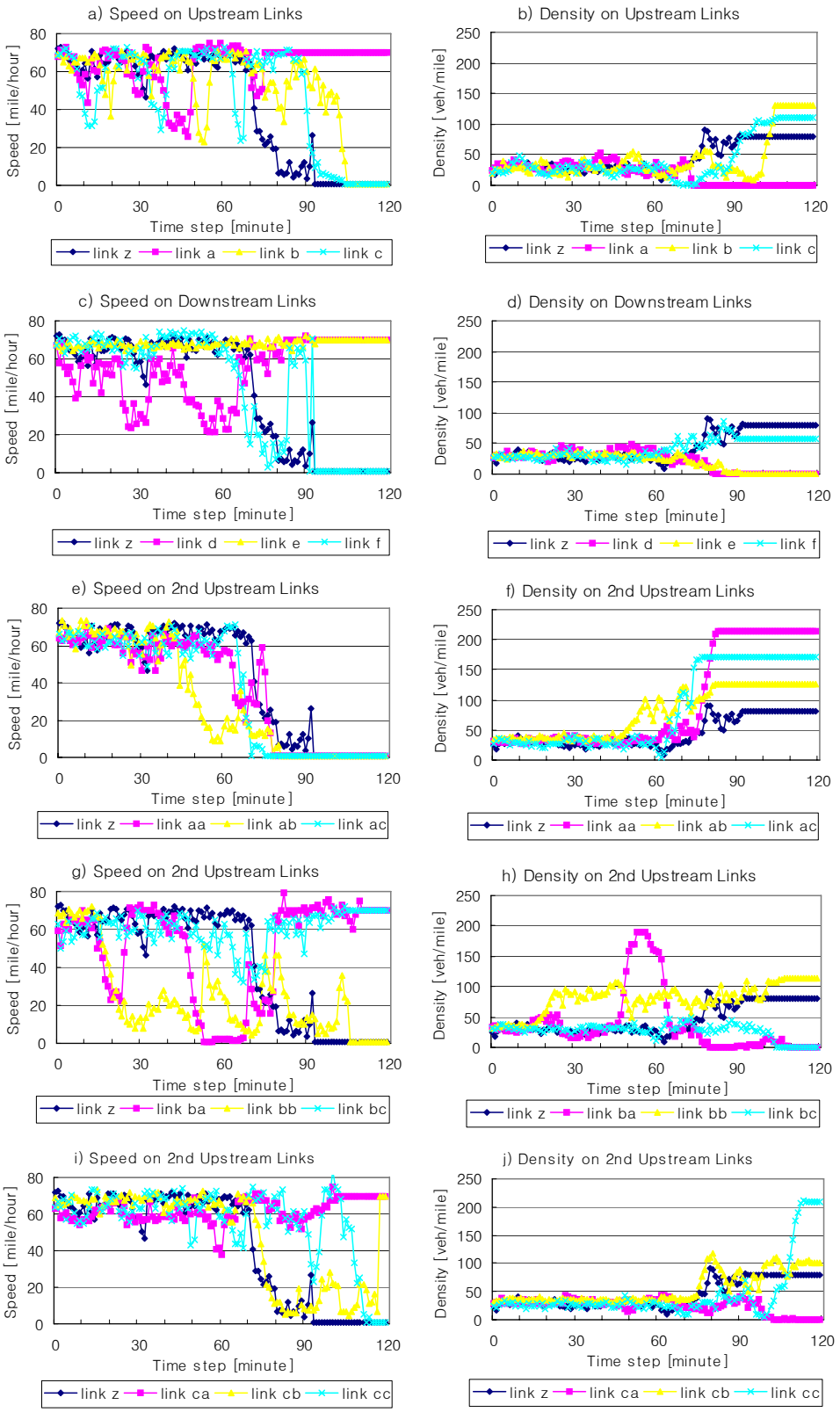


FIGURE 3-16 Speed and density on target link and neighboring links

The non-stationary conditions on all links are exhibited in Figure 3-16. This includes plots of speeds and densities on various links, each compared with the same statistic on the target link *z*. As shown in Figure 3-16, all links but link *e* are in an unstable traffic condition. Speeds can be seen to change suddenly. In particular, many links degraded significantly after time step 70 minutes and were, finally, jammed up. In the case of link *e*, congestion caused by spill-back traffic from interchange areas did not happen because vehicles got out of the network through that link. Increasing upstream traffic could make downstream traffic increase and cause traffic congestion. On the other hand, downstream congestion could spill back upstream and lead to a congested condition. In the absence of an incident, it is most reasonable that upstream traffic conditions at a given time will have an impact on downstream conditions at some later time, since many of the same vehicles will be involved. Thus, those correlations should be high.

3.2.2 Spatial and temporal relevance

This subsection describes spatial and temporal relevance of traffic information based on the results obtained from the simulation experiment in the previous subsection. On space mean speeds, 2-hour correlation, 15-minute correlation, and time correlation are discussed.

In order to analyze spatial relevance, the correlation on speed was used as a statistic. Figure 3-17 contains correlation coefficients on speed between each neighboring link and the target link over the whole simulation time (2 hours). In

Figure 3-17 the names of those links associate with each coefficient depicted are in parentheses.

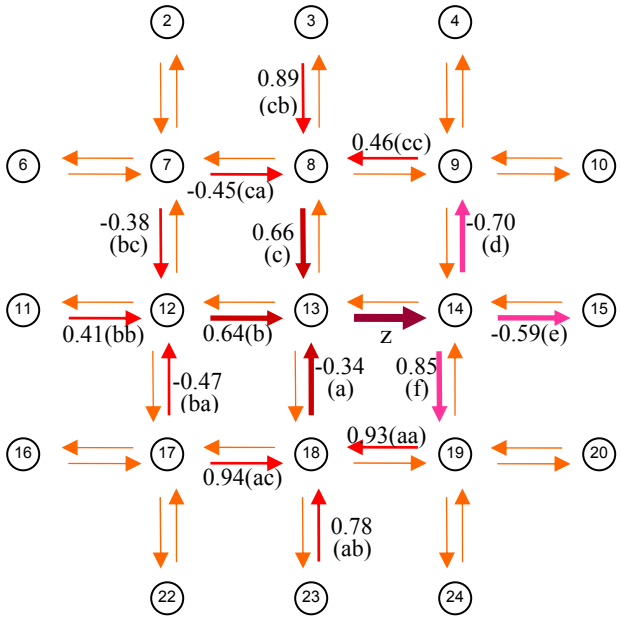


FIGURE 3-17 Two-hour correlation with speed on target link *z*

In Figure 3-17, each upstream link *a*, *b* and *c* of the target link *z* is correlated as much as -0.34, 0.64, and 0.66 respectively. For downstream results, the coefficients on links *d*, *e*, and *f* were -0.70, -0.59, and 0.85. These results suggest that speeds on links *b*, *c* and *f* are more related to that on link *z*. On several links (*a*, *d*, *e*, *ba*, *bc*, and *ca*), negative correlations were obtained. In comparing two link speeds, it would be difficult to conclude from a negative correlation of large magnitude such as link *d* (-0.7) that a "direct" causal relation exists between those links. For example, link *z* could be "indirectly" related to link *d* via other links. Certainly, it would be hard to argue that increasing the speed on any one link causes a decrease in speed on another link. Even though links *aa*, *ab*, and *ac* are one link farther away, high correlation

coefficients were obtained: 0.93, 0.78, and 0.94, respectively. However, it is believed that links *aa*, *ab* and *ac* have indirect relationships with link *z* in terms of speed because those links are connected to link *z* via link *a* with a negative correlation coefficient.

The correlations mentioned above attempt to capture the relations between links for the full 2 hours with one single value at the overall viewpoint. Figure 3-18 shows the variation of 15-minute correlation between each upstream/downstream link and the target link with density rate for maximum density (213 vehicles in this experiment). Each point was obtained through a sequence of 15 one-minute speed data; the first point was located at the 15th time step.

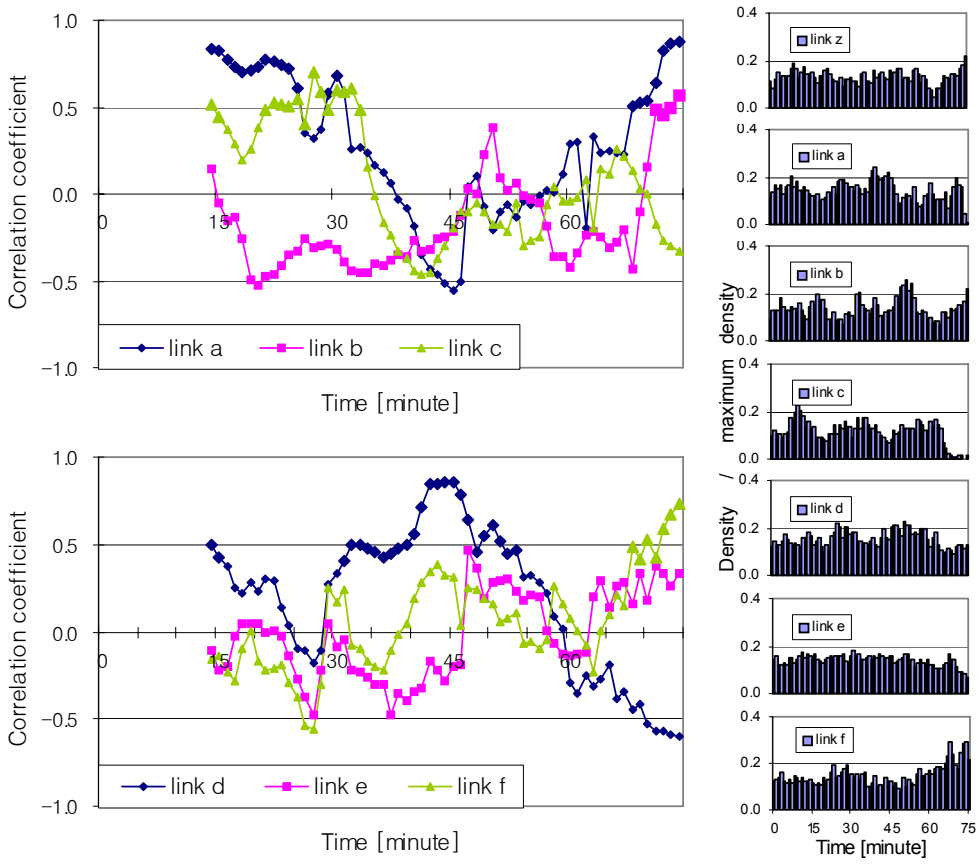


FIGURE 3-18 Correlation with speed on target link *z* for 15 minutes and density

As shown in Figure 3-18, unlike 2-hour correlations, the 15-minute correlation coefficients fluctuate significantly; no links are stable. For example, link *f* (0.85) was highly correlated to link *z* in 2-hour correlation, but reported negative value by 15 minutes and was not over 0.5 by 68 minutes. Link *a* was highly correlated to the target link by the first 10 minutes, link *d* during 32 to 54 minutes, and links *a*, *b*, and *f* from 68 minutes. High correlation coefficients (more than 0.4) were drawn with bigger markers. The right column of the Figure 3-18 shows density rate (density/maximum density) for each time step.

The temporal relevance of data on links located in the center area of the simulation network was investigated. Temporal relevance means the relation between present and past information. Speed data on chosen links were compared pair-wise, with a sequence of 1-minute time lags from the current time *t*. Figure 3-19 shows correlation coefficients by 30 time lags on a target link and neighboring links.

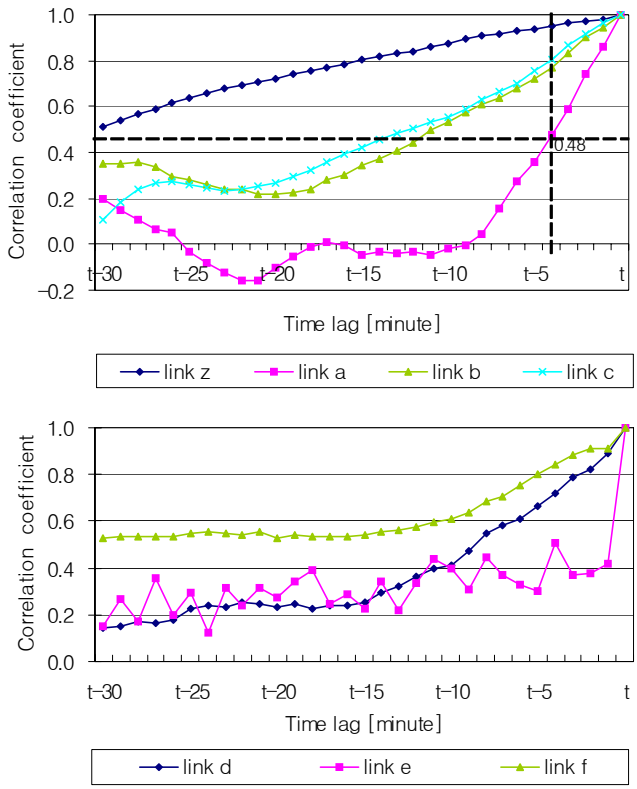


FIGURE 3-19 Temporal relevance

In Figure 3-19, the speed data for link *z* appeared to be temporally highly correlated, 0.81 at time t-15. However, links *a* and *e* were less correlated with past data: 0.48 only at four-minute time lag on link *a* and 0.3 at t-5 on link *e*. Links *b*, *c* and *d* showed analogous characteristics to each other; the coefficient curves for the three links follow a very similar pattern. The correlation coefficients at time lag t-5 were 0.72, 0.75 and 0.68, respectively.

As shown above, temporal relevance on traffic information appears though all links are not very correlated; correlations on some links were high with thirty-minute time lag and those on others were low even with five-minute time lag. Temporal

relevance of recent historic data is expected to be used for travel time expiration in vehicular ad hoc networks.

3.2.3 Linear model

In this subsection, a linear model using spatial relevance of traffic information is introduced. This model relies on the assumption that traffic data are strongly correlated with each other in terms of space. This is true in varying degrees for different network topologies. In general, spatially neighboring data are from upstream and downstream links. We expect these correlations to decline as space cells become more distant. Figure 3-20 depicts spatial relevance between a target link *z* and neighboring links *a*, *b*, *c*, *d*, *e* and *f*.

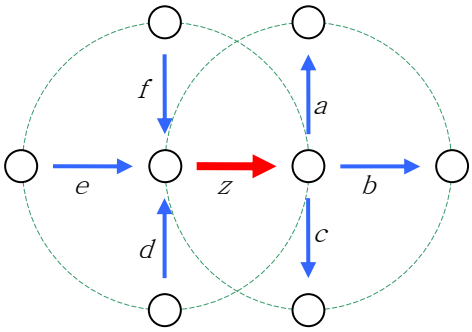


FIGURE 3-20 Spatial relevance example

In Figure 3-20, suppose that the travel time on Link *z* was missing. In order to estimate the missing travel time on Link *z*, we assume a linear relation with the neighboring downstream and upstream links, and generate a linear model.

$$\widehat{TT_t^z} = \alpha_0 + \alpha_a TT_t^a + \alpha_b TT_t^b + \alpha_c TT_t^c + \alpha_d TT_t^d + \alpha_e TT_t^e + \alpha_f TT_t^f \qquad (4)$$

where,

$\widehat{TT_t^z}$: Travel time estimate on Link *z* at time t

α : parameter

a, *b* and *c*: downstream links

d, *e* and *f* : upstream links

The goal, then, is to find, for this particular target link, the magnitude of the coefficients that should be used. In some cases, the inclusion of certain independent variables may not be statistically justified. The values of these coefficients may change with the particular traffic algorithm in mind.

A problem with unconstrained linear models is that they can provide impossible values that cannot happen in the real world, such as speeds of 150 mile/hour or negative speeds. This is more likely when only a small number of observations is available. Therefore, the results of the linear model should be followed by a process to adjust unreasonable estimates. The basic idea of the adjustment process in this study is to smooth estimates that are too high or too low as shown in Figure 3-21.

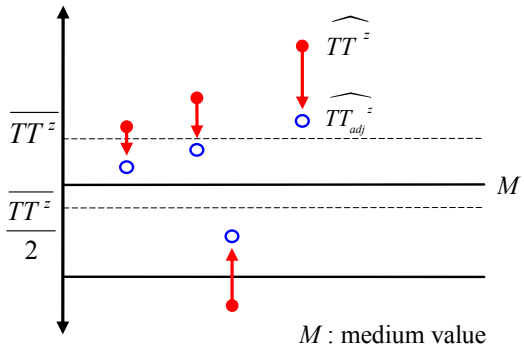


FIGURE 3-21 Concept of the excess adjustment

First of all, a medium value, M, assumed to be a neutral line between overestimates and underestimates, is defined. It would be in the middle of $\overline{TT}^z$ and $\overline{TT}^z/2$. In this study, M was chosen as a value two thirds of $\overline{TT}^z$. Next, all estimates from the linear model are shrunk (contracted) in the direction of the medium value. In particular, only the differences between $\widehat{TT}^z$ and M nonlinearly contract based on a shrinking factor defined as $\alpha(\widehat{TT}^z - M)^\beta$; 0.012 and 1.2 were applied for the values of α and β respectively in this paper. Finally, adjusted travel time is obtained through Equation (5).

$$\widehat{TT}^z_{adj} = \frac{\widehat{TT}^z - M}{\alpha(\widehat{TT}^z - M)^\beta + 1} + M \qquad (5)$$

If we compose a data structure based on the example in Figure 3-20, we can set up target link z with 3 downstream links (links a, b and c) and 3 upstream links (links

d, *e* and *f*). These are all links that are topologically adjacent to the target links. Figure 3-22 shows a sample data structure of the target link and the adjacent links.

	z	*a*	*b*	*c*	*d*	*e*	*f*	
t-15	56	72	65	67	23	12	58	
t-14	43	68	64	68	16	11	62	
.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	Coefficient estimation
.	.	.	.	.	.	.	.	
t-2	61	45	66	61	34	52	39	
t-1	45	51	63	59	49	34	37	
t	**?**	59	64	28	37	43	60	Missing data estimation

FIGURE 3-22 A sample of data set

In Figure 3-22, 15 consecutive data values were used as a unit of data set in order to estimate missing space mean speed. Coefficients for a linear model were estimated through those 15 consecutive data values (t-1~15), and the travel time (t) missed on Link *z* was estimated with coefficients obtained from previous time lags and data (t) on neighboring links at the same time window.

In order to evaluate this linear model, simulation results conducted in Subsection 3.2.1 were employed. A total of 120 one-minute observations of aggregated space mean speed were obtained. Of those, we kept the consecutive 68 pieces of data left after removing data such as jammed-up conditions (from the 69th time window, a traffic jam started to appear on some links). Figure 3-23 shows the virtual simulation road network used for this paper.

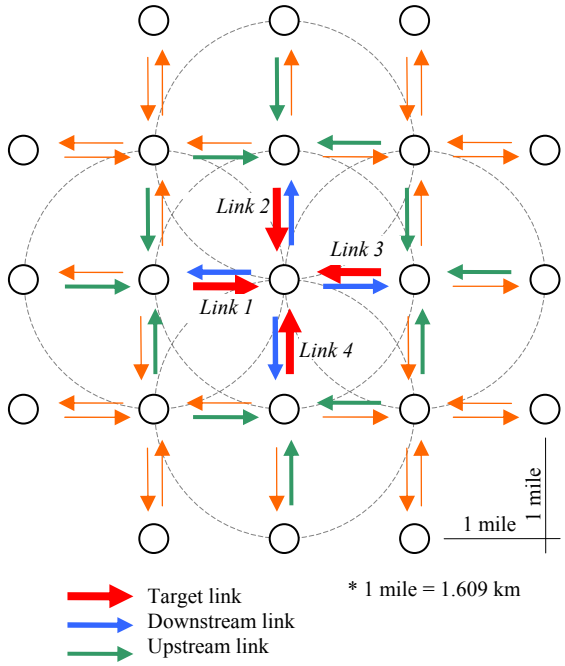


FIGURE 3-23 Network structure

We defined a target link and neighboring links on the simulation network, as indicated in Figure 3-23. The total of four links (links 1, 2, 3 and 4) were selected as a target. Four target links which are not affected by other characteristics such as no upstream links or downstream links were chosen.

TABLE 3-3 General observations

	Link 1	Link 2	Link 3	Link 4
# of samples	68	68	68	68
Maximum speed [miles/hour]	70.9	73.1	71.6	75.3
Minimum speed [miles/hour]	22.9	23.5	33.9	25.5
Median speed [miles/hour]	65.4	66.9	65.8	66.1
Average speed [miles/hour]	62.2	60.3	62.6	60.6
Standard deviation [miles/hour]	10.7	13.8	8.8	13.4

In Table 3-3, general observations for the four target links are summarized. Of 68 consecutive space mean speed data points, a total of 53 speed estimates per target link were provided because the first estimate needed 15 consecutive data points. The average space mean speeds for the four target links were more than 60 mile/hour; these links were not congested for most of the period. The differences between median and average speeds for links 2 and 4 (around 6 mile/hour) were higher compared to those for links 1 and 3 (3.2 mile/hour). Similarly, standard deviations for links 2 and 4 were higher than those from links 1 and 3, indicating a higher propensity for fluctuation on those links.

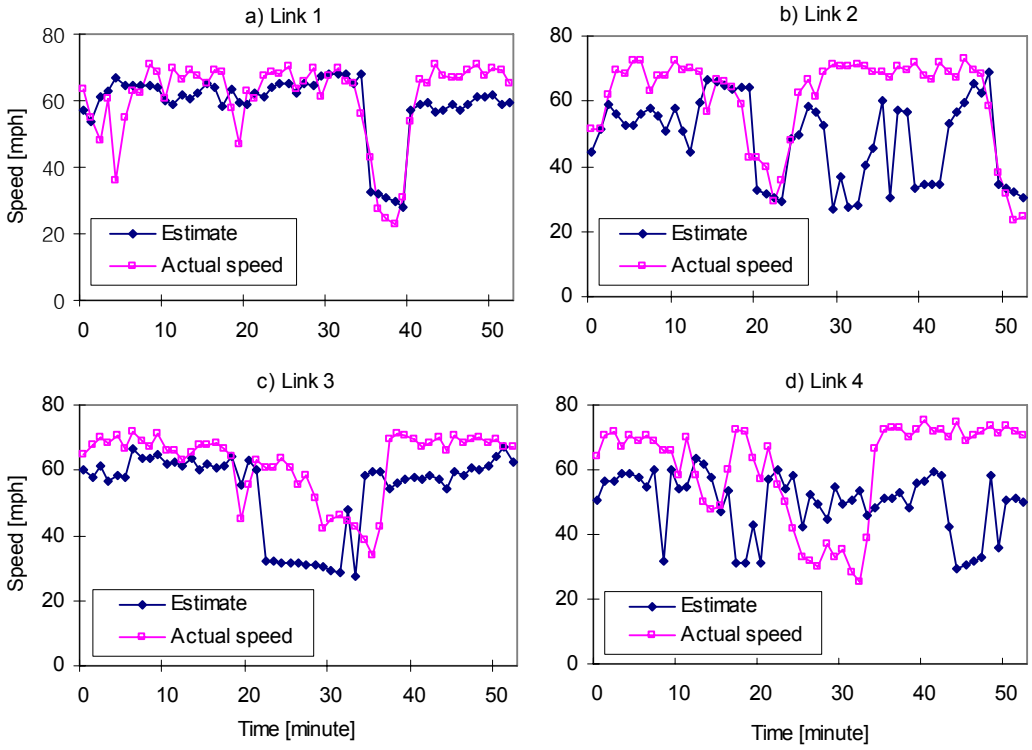


FIGURE 3-24 Estimated speed and actual speed

Figure 3-24 (a) shows that the speeds on Link 1 were the best, of those estimated through the linear model proposed in this paper. The match is best illustrated by the agreement between the simulator and predictor during the speed decrease that happened around the 39th time bin. In Figure 3-24 (b), actual speeds on Link 2 were estimated well only around time lags when the actual speed decreased, whereas large errors sometimes appear at high speeds. Figure 3-24 (c) shows good general agreement for the speeds on Link 3, although some deviations are present. The model generally performed poorly for Link 4, as shown in Figure 3-24 (d).

A linear model for predicting space mean speed on a target link using information from neighboring links was employed for imputing missing data. The results show that traffic conditions on the target link can be associated with those on neighboring links, and seem to be affected by compound relation of those links.

3.3 Discussion

This chapter discussed the patterns of individual travel times with aggregate travel times as a proxy of traffic condition and the data quantity for reliable travel time information based on individual travel time data obtained from a simulation experiment. Congested traffic conditions between uncongested and jammed conditions certainly changed and they showed high standard deviations. Large sample sizes (more than 23 in Figure 3-13) are required for reliable information in a congested traffic condition.

Section 3.2 showed spatial and temporal relevance amongst one-minute aggregated space mean speeds, a reciprocal of travel time. While spatial relevance

with neighboring links varied in a squared road network, temporal relevance at the same link seemed to be associated amongst recent historic data. In some links, however, it became quickly stale. In temporal relevance degradation of travel time, correlation coefficients of travel time on many links fell down under 0.5 over a period of 14 minutes (Figure 3-19). These results for travel time are used to determine the degradation of travel time information in a framework design.

The next chapter introduces how this study builds a simulation framework for VANET applications. The simulation framework integrated with transportation and communication simulators is designed based on an information model in VANET applications which is defined in this study and implementation techniques are described.

Chapter 4: Simulation Framework

Research issues related to inter-vehicle communications for transportation applications are extraordinarily complex, with numerous complicated and interdependent stochastic inputs. As a result, the only reasonable method to evaluate ideas for real systems is through simulation. These ideas are far enough ahead of the curve, however, that no single current commercial package offers the full range of features required to study these systems. An integral component of the work for this dissertation, therefore, was the development of an integrated transportation and communication simulation framework, having as its target a wide range of applications to VANETs. Transportation and communications are essentially stove-piped disciplines, so the only way to build an effective simulator for both simultaneously was to start with the most appropriate simulator within each domain and then to integrate. The definition of "most appropriate" included both well-known performance within the domain and an ample Application Programming Interface (API) with which to override default behavior and build links to external functions in real time. The most critical areas of interoperability between the two simulators were time management and mobility management.

This chapter describes the development of the integrated simulation environment. Section 4.1 begins with a description of the simulation requirements, including the information model proposed for the dissertation, as well as the desired architecture. Section 4.2 describes some state-of-the-art simulators in the transportation and communications domains and shows how the choices of Paramics (transportation)

and Qualnet (communications) were arrived at. This section also shows how the integrated simulation platform was implemented, including the issues of time and mobility management mentioned above. The section closes with a specific description of the inter-vehicle communications mechanisms designed and implemented in this simulation platform for the purposes of this dissertation.

4.1 Simulation design

One of the most fundamental changes that can be expected from VANETs used for transportation management purposes is that the nature of the information involved in the decision-making will change. The existing traffic data paradigms are well-established, arguably entrenched, and will need to be re-thought completely in order to best exploit this evolution of technology. Because existing simulation tools were built while the old paradigms were active, one has to be at least suspicious that they may not be directly extensible to this new environment. Thus, the first important task to be conducted as part of this research was to develop an information model that describes the types, frequencies, etc. of the information packets that are expected to be used by models supported by VANETs, as well as their collection and distribution mechanisms.

With this information, it was then possible to design a cooperative architecture expected of the two domain simulators. Specific details depend on the simulators chosen, of course, but the general guidelines of the interfaces, data dictionary, etc., were determined at this point. The following two subsections describe the

development of the information model and simulation platform architecture in more detail.

4.1.1 Information model

In order to understand the target system of the proposed simulation framework, an information model in transportation systems based on VANETs is presented in this study. In the information model, various traffic events – such as vehicle movements – are collected as data, in some manner, from the transportation systems and are then fed back to the transportation system according to some collection of preset logic. The information model considers traffic events as information (thereby assuming that the means to collect the information accurately are available) and processes the information to decide reactions. The model presented here defines procedures by which information is collected, disseminated, utilized and fed back in a VANET-based transportation system. Figure 4-1 shows a diagram of the information model which this study focuses on.

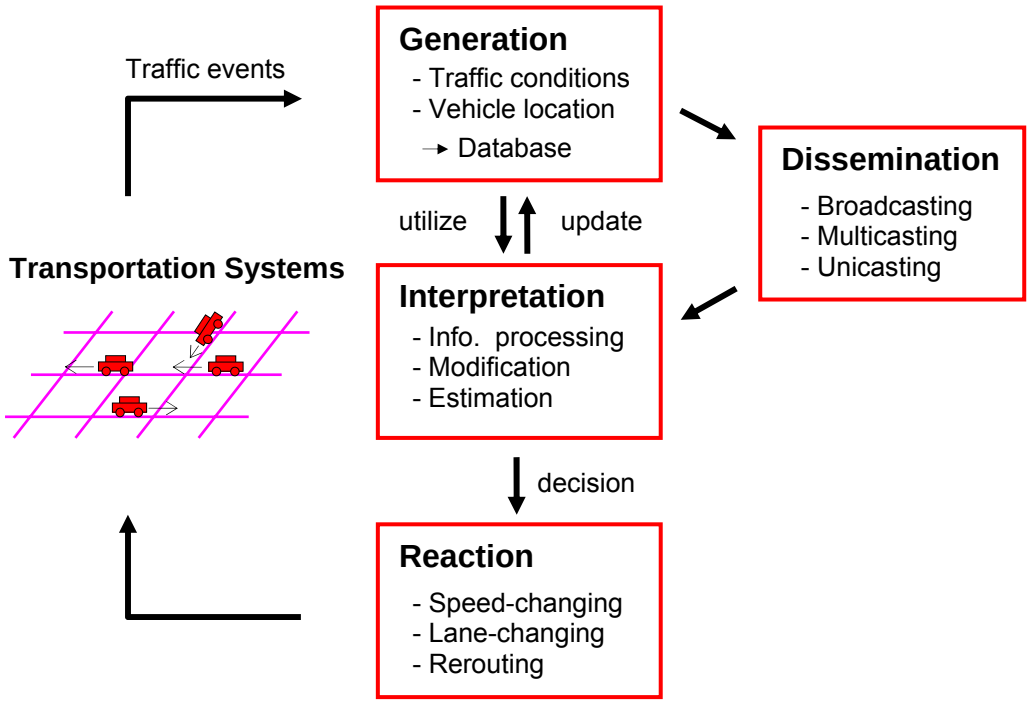


FIGURE 4-1 Information model for VANETs

The proposed information model for VANETs consists of four stages: generation, dissemination, interpretation and reaction. To support VANETs, the first stage of the information model is the generation of information which describes traffic states (conditions) for transportation mobility purposes, and vehicle locations for transportation safety purposes. Since traffic information is being collected by individual vehicles, at this point the state information must be that which can be estimated by a single variable. For example, the travel time of a single vehicle over a given link can be used as an estimator of the average travel time for many vehicles over that link (which might be a meaningful macroscopic definition of state for that link), but a single vehicle cannot characterize the flow on a link.

Vehicles are assumed to be equipped with navigation / location hardware with sufficient spatial and temporal resolution for the intended applications. Thus, a vehicle can determine and report its own position. Meaningful information, the definition of which depends on the intended applications, would be generated by any events that can be sensed by the vehicle, most notably vehicle's own movements in the transportation system. As described in Section 1.1 of Chapter 1, it might be most efficient for vehicles to have a collective sense of "the norm" for traffic states at any given point in time and space, and only to report on conditions sufficiently outside of that norm. Where it is appropriate to assume that no links would be entirely devoid of communications-equipped vehicles, this structure allows the total absence of data to also be interpreted as information. Any information generated in the vehicle is accumulated in internal information bases such as a database inside an Onboard Unit (OBU).

Information generated is disseminated to appropriate recipients at the second stage. Depending on the applications, information would be destined either to specific vehicles (unicasting or multicasting) or to all other vehicles within transmission range (broadcasting). Since mobility in VANETs is extremely high and nodes are essentially anonymous, most applications will (or should) aim to deliver information to all other vehicles within range. An exception to this would be applications should as mobile web, but this dissertation is focused only on transportation management applications. The simplest method for every vehicle to send information to every other vehicle within range would be periodic broadcasting, although various more complicated broadcasting schemes should be studied for

different applications. For better performance, the broadcasting interval can be adjusted according to the situation.

When a vehicle receives new information from other vehicles, it would either discard or assimilate the information, depending on its interpretation. At the stage of interpretation, procedures for how to understand the new information and what to do with it are conducted. The interpretation may cause an update or modification to a vehicle's own information base with new information.

At the final stage, accumulated information should result in some reactions from vehicles (or their drivers) such as speed changing, lane changing, or rerouting. The reaction defines rules for such responses to new information. This reaction usually feeds back to the transportation system, possibly generating new traffic situations.

This information model can cover most applications of VANETs which base the individual vehicle's reaction to the transportation system on collecting, processing, and disseminating traffic events. The simulation framework is designed according to the presented model and, therefore, most VANET-based transportation system can be properly simulated within this framework.

4.1.2 Simulation framework design

This section describes the simulation framework designed to work with the information model presented in the previous section. The simulation framework was designed with the notion that a simulator for VANETs should provide: 1) simulation for the transportation system, 2) simulation for the vehicular ad hoc network, and 3) application logic for intelligent systems. A transportation simulator models a

transportation system by simulating the behavior of vehicles according to internal traffic models such as car following, lane changing, shock waves, and queuing. A communications simulator demonstrates various aspects of wireless ad hoc networks such as the underlying radio channel and multiple access control. The application logic for VANETs presents how the system generates, disseminates, and interprets the information. It also defines each vehicle's reaction to the information it receives. To provide a simulation framework for VANETs, the transportation simulator and the communication simulator are integrated by creating an interface between the two. The simulation time and vehicle positions must be synchronized in the two simulators via inter-simulator communication. Figure 4-2 shows the architecture of the simulation engine, including the transportation and communications components, together with the communications infrastructure that joins them.

To synchronize the simulation time between both simulators, they exchange current simulation times at a given precision. Since the communication simulator does not support vehicles' movements, the transportation simulator periodically provides the communication simulator with vehicles' locations. Upon occurrence of traffic events, the transportation simulator generates data, decides on the intended recipients, and lets the communication simulator represent inter-vehicle communications. In this study, it is assumed that the system disseminates information by periodic broadcasting (this constraint is intended for the case study presented in later sections). The framework, however, can accommodate more intricate communication schemes such as adaptive broadcasting (Wischhof *et al*., 2007 and Saito *et al*., 2007). The communication simulator informs the

transportation simulator about data receptions, and the transportation simulator then interprets the information, and modifies it and reacts if needed. The transportation simulator and communication simulator proceed independently (as different processes), but each with constraints imposed by the other. In particular, each has the ability, through the API, to suspend execution of the other long enough for the non-native data to be updated appropriately. When this synchronization scheme is executed with sufficient resolution, it has the effect of mimicking a combined simulation environment.

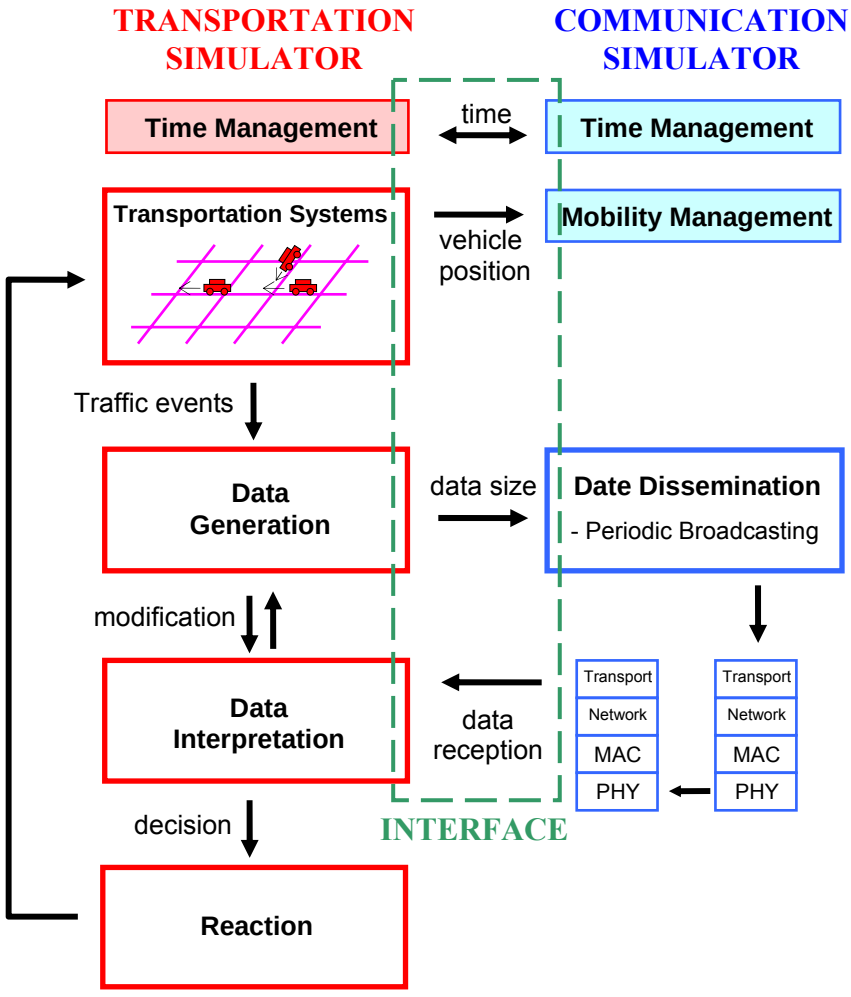


FIGURE 4-2 Framework implementation

4.2 Implementation

In this section, the specific modes to implement the simulation framework designed in the previous section were described. Proper tools for transportation and communication simulations were chosen, and these two simulators were synchronized with respect to simulation time and node locations. The last subsection shows data communications between the simulators with data format from QualNet to Paramics for updates of traffic data via intervehicle communications.

4.2.1 Simulation tools

In order to simulate vehicles' mobility in transportation systems and ad hoc networking among vehicles, simulators oriented to these specific purposes are employed. For these purposes, the transportation simulator should be a microscopic simulator capable of describing correlated movements of individual vehicles. The communication simulator should simulate the 7 layers in the open systems interconnection (OSI) reference model proposed by the International Organization for Standardization (ISO), and should be able to handle large communication networks composed of equipped vehicles.

Corsim (Corsim homepage, 2007), VISSIM (VISSIM homepage, 2007), AIMSUM (AIMSUM homepage, 2007), and Paramics are well-known microscopic transportation simulators. Of those simulators, Paramics version 5.2, developed by Quadstone Paramics, is employed in this study as the transportation simulator. Paramics is a microscopic traffic simulation tool producing movements and behavior of each individual vehicle. It can be used to replicate traffic on a wide variety of

transportation networks. Paramics' primary flexibility is that it allows users to access its internal mechanisms via a convenient Application Programming Interfaces (API) (Paramics homepage, 2007).

Most sources in the literature in mobile wireless networks use NS-2 (NS-2 homepage, 2007), OPNET (OPNET homepage, 2007), GloMoSim (GloMoSim homepage, 2007), or QualNet for evaluation tools. In this study, QualNet version 4 was chosen as the communication simulator. QualNet, developed by Scalable Networks Technologies Inc., is the commercial successor to GloMoSim. QualNet can simulate large scale wireless networks as a packet level simulator for wired and wireless networks. For example, Scalable Networks Technologies describes QualNet by saying that it can simulate a communication network with thousands of nodes with reasonable performance due to improvements in design such as parallel execution and smart memory management. QualNet supports all seven layers from the physical layer to the application layer in the OSI reference model. On the wireless physical layer, protocols 802.11 DCF/PCF, 802.11 a/b/g/e and 802.16(e) are supported (QualNet homepage, 2007).

4.2.2 Mobility management

In this simulation framework, the locations of nodes (vehicles) in QualNet are synchronized with those of equipped vehicles in Paramics. Paramics was programmed to periodically send to QualNet the positions of all equipped vehicles currently active in the system with a timestamp (Paramics can control the time step to

within 10 milliseconds). Figure 4-3 shows how these simulators synchronize vehicles' movements and graphically illustrates the expected error from this method.

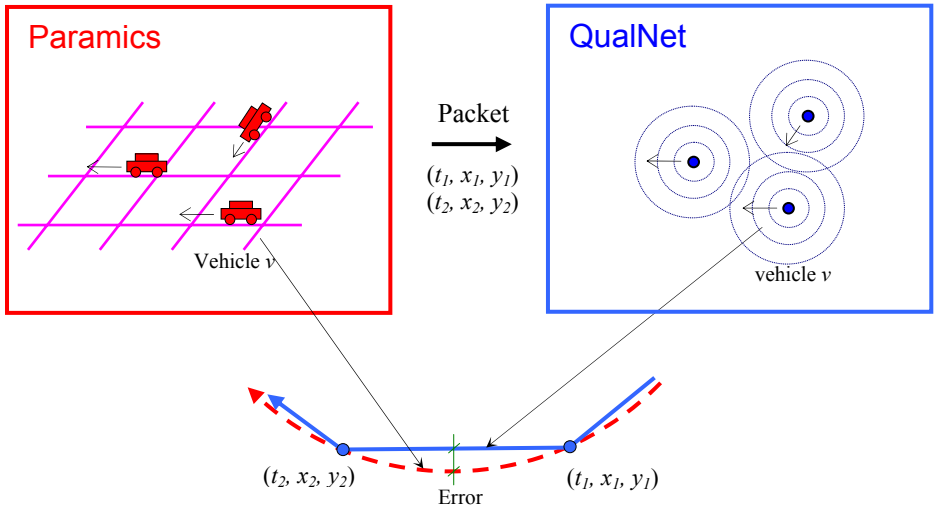


FIGURE 4-3 Movement synchronization and expected error

As shown in Figure 4-3, QualNet moves vehicle (node) positions along a linear path with time-stamped vehicle positions. For example, if QualNet received (v, t_1, x_1, y_1) and (v, t_2, x_2, y_2) at the next period from Paramics, where $t_2 > t_1$, then the vehicle v is assumed to depart from location (x_1, y_1) at time t_1 and arrive at location (x_2, y_2) at time t_2 and to have done so along the straight line between (x_1, y_1) to (x_2, y_2). This "interpolated mobility" could cause incorrect vehicular positions if the actual vehicle trajectories are non-linear. This error, however, can be made negligible, since highways have bounded curvature and the update time for the simulation is quite small. Figure 4-4 contains the format of the packet transferred from Paramics to QualNet.

```
{
      float   time;       // current time in Paramics
      int     vcnt;       // number of vehicles in this packet
      {
            int     vid   // vehicle id
            float    x    // x coordinate
            float    y    // y coordinate
            int    size   // size of data in the vehicle
      } // 1
      {
          ...
      } // 2
           .
           .
           .
      {
          ...
      } // vcnt
}
```

FIGURE 4-4 Packet format from Paramics to QualNet

In Figure 4-4, the first data indicates the current time that Paramics sends to QualNet, and is followed by a total number of vehicles contained in the packet. Now, data describing locations for each vehicle are attached: vehicle ID, *x* coordinate, *y* coordinate, and data size which each vehicle stores at that time. About data size, it will be more discussed later.

4.2.3 Time management

QualNet takes the form of Discrete Event Simulation (DES) software, the most widely used form in communication simulation. In DES, the state of the system is assumed to change only at discrete epochs; in other words, the simulation time (or simulation clock) proceeds only when an event happens (e.g., 1, 5, 6, 20, ...) rather than increasing by constant time units (e.g., seconds 1, 2, 3, 4, …). The former is also

called “event driven” and the latter called “time driven.” Since QualNet is an event-driven simulator and Paramics is a time-driven one, it is not a trivial task to keep the clocks of these two simulators synchronized. Figure 4-5 shows two different cases where the simulators have slightly different impressions of the simulation time, and the ramifications thereof.

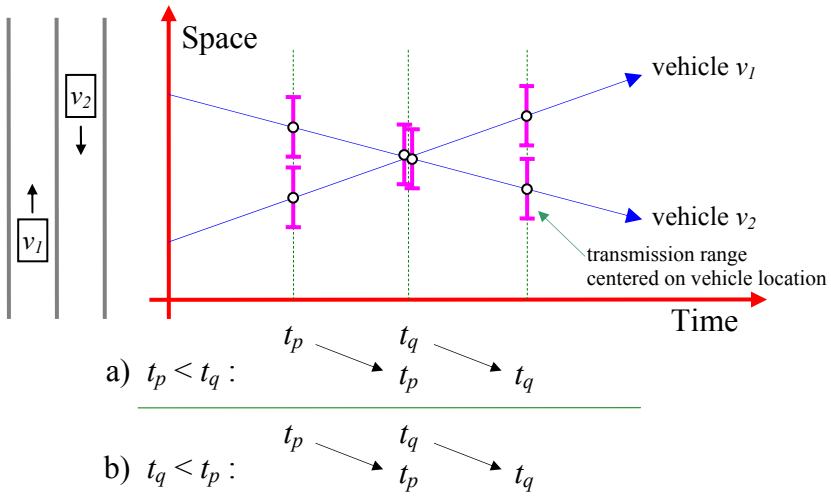


FIGURE 4-5 Two cases by different simulation time

As Figure 4-5 a) shows, if QualNet time (t_q) is ahead of Paramics time (t_p), then QualNet is performing communications with incorrect (or delayed) vehicular positions. With high-speed vehicles, even a small delay ($t_q - t_p$) can cause large errors in vehicle positions. Incorrect vehicle positions can cause data losses and unnecessary data receptions, which should not happen with correct positions. This study considers this error critical because data losses and unnecessary data receptions cannot be recovered. If, on the other hand, Paramics time is ahead of QualNet time (Figure 4-5 b), QualNet is always aware of exact vehicle positions while Paramics is

not aware of some data receptions that happened between t_q and t_p until QualNet time reaches t_p. Thus, the arrival of data from other vehicles can be delayed by up to $| t_p - t_q |$. Such a delay might impact the vehicle's reaction in response to the data. With a small value of $| t_p - t_q |$, i.e., 1 second, this error minor is considered, if not negligible, because reception of a few data packets (or delays thereof) is unlikely to cause a significant change in a vehicle's reaction during such a short period of time. Figure 4-6 shows graphically how to synchronize the simulation time of the simulators in this study.

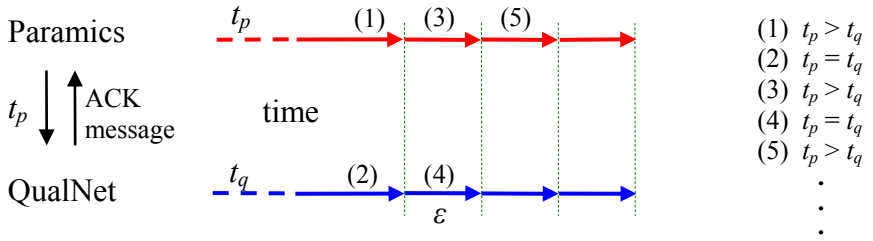


FIGURE 4-6 Synchronization of Paramics and QualNet

Given a maximum time error ε as shown in Figure 4-6, the Paramics time is always kept ahead of QualNet time, by at most ε: $0 < t_p - t_q \leq \varepsilon$. This synchronization method guarantees that the delay of data delivery can be constrained above by the parameter ε. Paramics periodically sends the current simulation time t_p to QualNet every ε seconds. QualNet sends an acknowledgement message to Paramics in order to inform Paramics that QualNet has processed all events happening before the current Paramics time t_p. Since QualNet is a discrete event simulator, QualNet processes events until t_p and stops if the time of the next event to process is later than

t_p. Assuming there is at least one event in each ε period, the time difference between these simulators is always less than ε. In the unlikely event that this was to be both violated and important, one could always synthesize events in Paramics via innocuous functions in the Paramics API; therefore this performance standard can be guaranteed.

4.2.4 Intervehicle communication

In the simulation environment, synchronized in terms of time and mobility, equipped vehicles communicate, which means transmitting data. QualNet simulates the vehicles' broadcasts of their data via an ad hoc network, and sends Paramics the results of the broadcasts to update traffic data based on the broadcasting time. Figure 4-7 contains packet format from QualNet to Paramics.

In QualNet, data reception by broadcasting induces events. As shown in Figure 4-7, QualNet notifies Paramics of data reception by sending a list of (*rvid*, *size*, *svid*, *rtime*) indicating "a receiving vehicle *rvid* receives a data packet of *size* from sending vehicle *svid* at time *rtime*." In this list of data, QualNet notifies Paramics only of the size of the data broadcast in QualNet unlike Wu's (2005) simulation framework in which the whole data broadcast are sent from a communications simulator to a transportation simulator. This simplification improves the simulation performance by decreasing the communications loads between the simulators. The results of data transmission/reception are attached to periodic time-synchronization packets mentioned in Section 4.2.2 to reduce the communication overhead.

```
{
    float   time;       // current time in QualNet
    int     vcnt;       // number of vehicles in this packet
    {
        int  rvid;    // vehicle receiving data
        int  rcnt;    // number of data received
        {
            int      svid;    // vehicle sending data
            float    rtime;   // received time
            int      size;    // data size of sending vehicle
        } // 1
        {
            ...
        } // 2
            .
            .
            .
        {
            ...
        } // rcnt
    } // 1
    {
       ...
    } // 2
        .
        .
        .
    {
        ...
    } // vcnt
}
```

FIGURE 4-7 Packet format from QualNet to Paramics

4.3 Discussion

This chapter described how to design and implement the simulation framework which this dissertation develops. In particular, it depicted how to integrate two different simulators (Paramics and QualNet) in detail; Paramics is time-driven and QualNet is event-driven. This framework was designed with a variable error tolerance *e*, a maximum time error, which would be determined according to applications: e.g., 1 second for traffic information systems and 0.1 seconds for

collision warning systems. While this research focused on a traffic information system, it could be applied to transportation safety systems as well depending on a determined error tolerance.

The next chapter contains how the simulation framework implemented in this chapter is applied to a traffic information system. The first section shows how a traffic information system based on a VANET is composed and collects travel time information. The second section describes how the simulation framework works in detail.

Chapter 5: Traffic Information System Application

In this chapter, the integrated simulation framework developed in this study is applied to a traffic information system in which vehicles are provided traffic information through intervehicle communications. The system configuration for a traffic information system based on a VANET is introduced, and the simulation model to apply to that system is described specifically.

5.1 Traffic information system configuration

This section introduces a VANET-based traffic information system which this research envisions in the real environment. The overall system configuration, the process of generating self-recorded travel times, and the internal processing in an onboard unit are described.

When a traffic information system based on a VANET is deployed, we distinguish between vehicles involved in this system and ones not involved. The involved vehicles possess communications hardware and onboard units, and are referred to as equipped vehicles. Unequipped vehicles are also present, but are not able to participate in the data collection or generation. This is a major distinction from other kinds of sensing technology (e.g., inductive loop detectors) where all vehicles contribute to the generation of data, but only equipped vehicles can receive information about those data via wireless communications. It is assumed that each vehicle equipped with an onboard unit will also have location technology equivalent to differentially-corrected GPS, as well as a digital map database and supporting

software and database tools. Presumably, communications software would also reside in the same housing, although this is irrelevant at the conceptual level. Figure 5-1 illustrates the overall system configuration of a traffic information system based on a VANET.

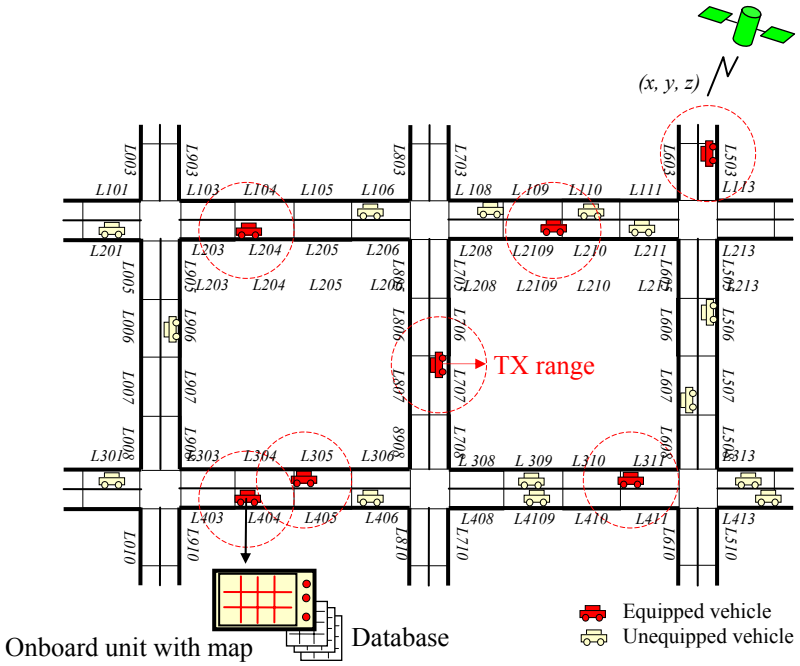
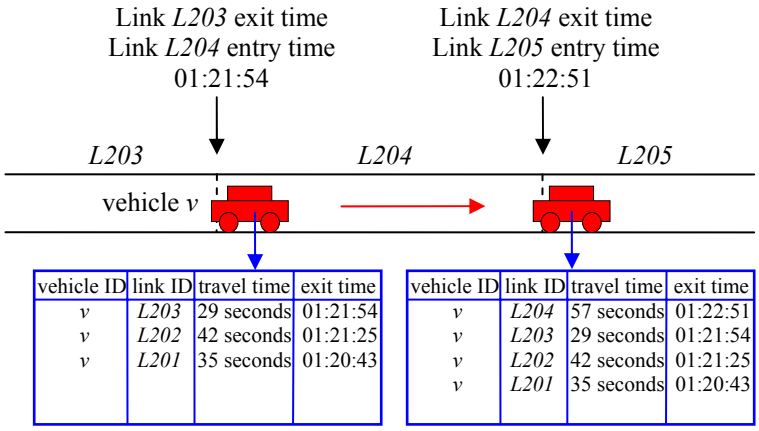


FIGURE 5-1 Traffic information system based on a VANET

In Figure 5-1, vehicles equipped with an onboard unit travel on the road network intermingled with unequipped vehicles. Equipped vehicles can recognize their current locations through location information such as longitudes and latitudes from satellites. In particular, a reference road map included in an onboard unit allows a vehicle to learn which link they are on. The reference road map defines start and end locations for all links with an associated link ID such as *L309*, *L204*, etc. Traveling on the road network, equipped vehicles transmit travel time data packets, which they store in their database, to other equipped vehicles within transmission range, and they

receive what other vehicles transmit. Figure 5-2 illustrates how travel time data are generated.



vehicle ID	link ID	travel time	exit time
v	*L203*	29 seconds	01:21:54
v	*L202*	42 seconds	01:21:25
v	*L201*	35 seconds	01:20:43

vehicle ID	link ID	travel time	exit time
v	*L204*	57 seconds	01:22:51
v	*L203*	29 seconds	01:21:54
v	*L202*	42 seconds	01:21:25
v	*L201*	35 seconds	01:20:43

FIGURE 5-2 Example of map-based travel time generation

In Figure 5-2, when an equipped vehicle *v* travels on the roadway, the vehicle recognizes its current location using location information obtained from GPS. Since it is assumed that all equipped vehicles use the same reference map in their onboard units, they know link information such as link location, distance, the number of lanes, etc., and key all of this information to link IDs. Presumably, the onboard maps could be kept both consistent and up-to-date via communications from map servers located throughout the network, using the communications equipment already in the equipped vehicles. Returning to the figure, based on the reference map in the onboard unit, the vehicle obtains its own travel time of link *L204*, the difference (57 seconds) between the exit time (01:22:51) and the entry time (01:21:54), when it leaves the link. In the same way as Figure 5-2, equipped vehicles record their own travel time data with the

vehicle ID, the link ID, and the exit time. Figure 5-3 shows how equipped vehicles exchange travel time data with each other.

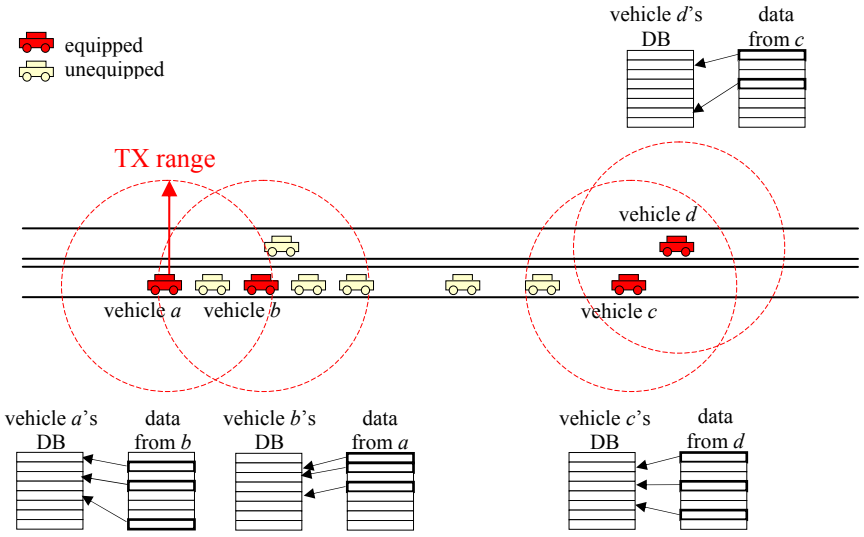


FIGURE 5-3 Example of travel time data exchange

Figure 5-3 shows two cases: data exchanges between equipped vehicles traveling in the same direction and between vehicles traveling in opposite directions. Equipped vehicles *a* and *b* traveling in the same direction within the transmission range communicate and exchange travel time data which they each have. Of data obtained from vehicle *b*, vehicle *a* selects and updates only travel time data which its own database does not have, and vice versa. Equipped vehicles *c* and *d* traveling in opposite directions also communicate and exchange travel time data since they are within the transmission range of each other. In that case, vehicle *d* is expected to convey data from vehicle *c* as well as its own data to vehicles *a* and *b*. Figure 5-4 depicts how an onboard unit works internally.

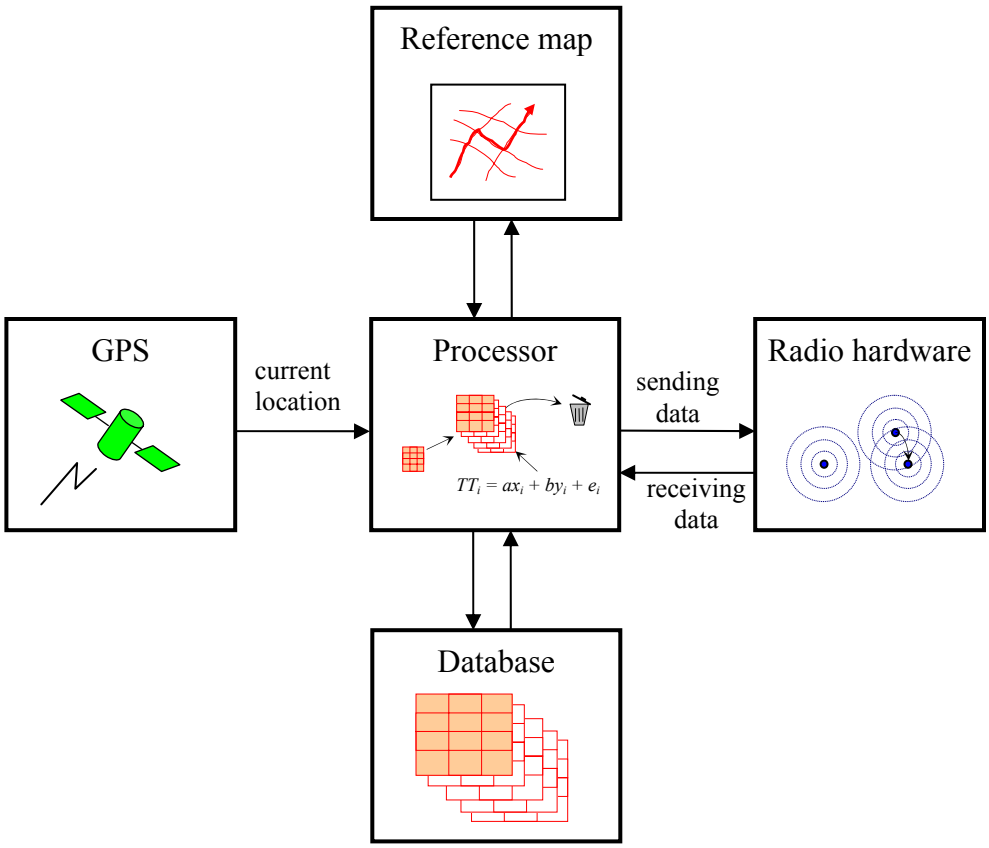


FIGURE 5-4 Internal configuration of onboard units

In Figure 5-4, an onboard unit consists of a reference map, GPS, radio hardware, database, and processor. Receiving current location information from satellites, GPS allows vehicles to learn where they are on the reference map. Through the radio hardware, equipped vehicles transmit and receive travel time data. All of these internal components are connected with the processor. The processor interprets travel time data received from other vehicles, and determines whether to store or throw away the data. For dynamic routing, the processor finds the shortest path based on travel time stored in database. Finally, equipped vehicles reroute to avoid congestion.

5.2 Simulation model architecture

In this section, the simulation model architecture for the traffic information system described in the previous section is introduced and specific methodologies are mentioned. Figure 5-5 shows the logic of each stage of the traffic information system application which this section describes.

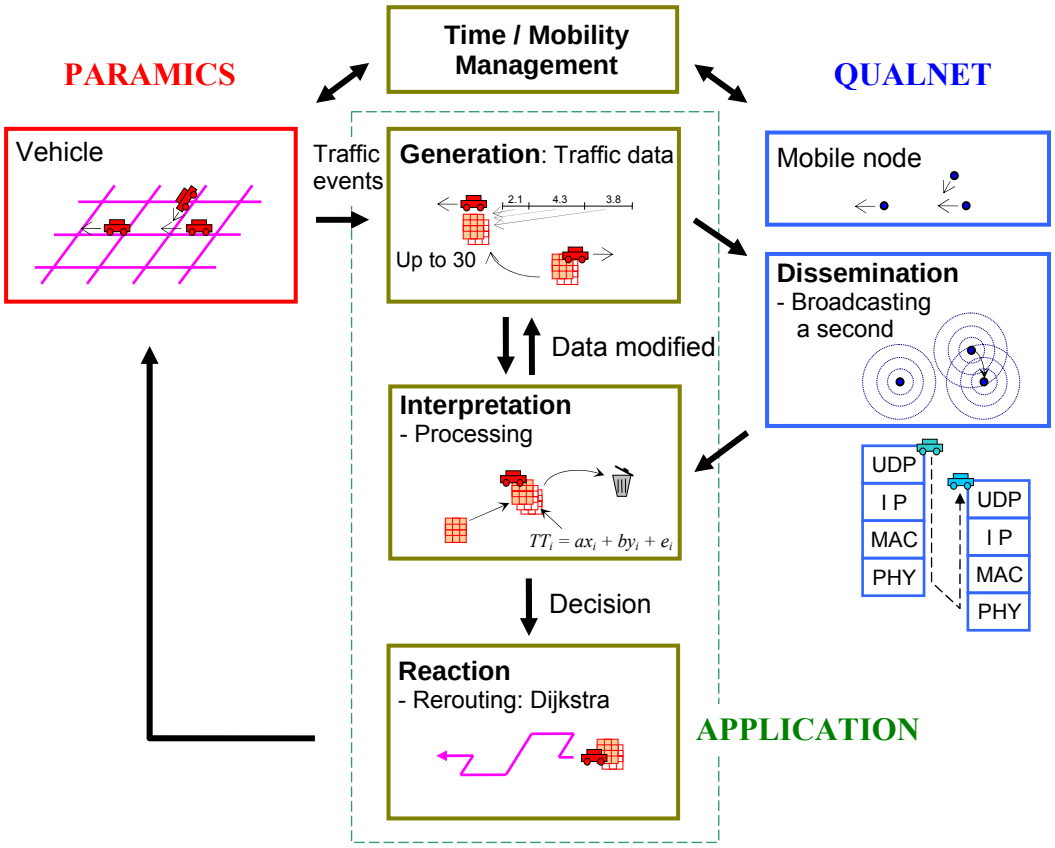


FIGURE 5-5 Traffic information system application

As shown in Figure 5-5, the basic logic is same as the real system described in the previous section. Travel time data are generated by equipped vehicles and stored in their database. Through broadcasting, vehicles share travel time data which they

have in their database. At the Interpretation stage, the delivered travel time data are processed, which might include assimilation into an existing data set, discarding of stale data, modification, and calculation. Travel time data in an on-board unit are used to conduct dynamic routing. Rerouting is conducted as a reaction in this simulation model, and the results make traffic conditions change. Based on Figure 5-5, the following subsections specifically describe how each stage is simulated.

5.2.1 Vehicle release and travel time generation

In the simulation, vehicles are released from certain ends of links, called "zones." When vehicles enter the road network, their origin and destination zones are determined. The first process in the simulation releases vehicles into the network, depending on the origin and destination matrix in which the release rates, the number of vehicles per time period (i.e., 1500 vehicle/hour), are defined. Vehicles are released according to a release algorithm which uses a number generated from a uniform random distribution to determine the headway between released vehicles.

In this simulation framework, a vehicle is determined to be equipped or not on the basis of a Bernoulli random variable sampled for each vehicle upon its entrance to the network, with a parameter equal to the intended market penetration rate. Paramics provides the API function "`qpx_VHC_release ( )`" which is called when a vehicle is released from a zone (see code in Appendix A). By implementing this function, we can create custom data fields for each vehicle, stored in the user-definable "userdata" structure provided by Paramics. For this simulation, we defined six fields in the vehicle userdata structure.

```
struct VHC_USERDATA_s
{  int VHCID;       // vehicle name
   Bool equipped;   // equipped or not
   float ReleaseTime;
   float EntryTime;
   struct VHC_TTDB_s *db;
   int DataSize; };
```

FIGURE 5-6 Vehicle userdata structure

In Figure 5-6, "VHCID," the first field of the structure, is a unique name provided for every vehicle, both equipped and unequipped. This ID tag is the means by which any vehicle's information can be accessed via the API. The second field, "equipped," records whether or not the vehicle participates in the VANET-based traffic information system, and this field is set as described above. At the same time, the vehicle's "ReleaseTime" is filled in according to the current simulation clock. To calculate link travel time, "EntryTime" is temporarily stored whenever the vehicle enters a link, as illustrated in Figure 5-2. "VHC_TTDB_s" is a pointer indicating the vehicle's database in which travel time data are stored, and the total size of data stored in the database is updated in "DataSize" whenever the database is updated.

To generate travel time data, the entry time and the exit time of links are used. The Paramics API function "`qpx_VHC_transfer`" is called whenever a vehicle traverses a node. In this function, the entry time of the vehicle is recorded in the userdata structure, and the travel time is calculated from the difference between the entry time and the exit time of the link when the vehicle leaves the link (see code in

Appendix A). Figure 5-7 shows the travel time data packet structure stored in database.

```
struct TravelTime_s
{  int VHCID;          // who measured travel time
   int LinkIndex;      // where travel time was measured
   float TravelTime;   // value
   float ExitTime; }; // when travel time was measured
```

FIGURE 5-7 Travel time data packet structure

As shown in Figure 5-7, a single travel time data packet consists of 4 fields in this framework. It can be, however, different depending on an application target. For example, location data such as a longitude and latitude could be included in the data packet in VANET applications for transportation safety.

5.2.2 Data dissemination

This research assumes that this traffic information system disseminates travel time data packets by simple periodic broadcast. More advanced communications schemes could also be considered, such as adaptive broadcast which changes broadcast intervals to reduce communication collisions. In particular, such schemes would be appropriate to optimize communications performance in high density traffic conditions with a high market penetration rate.

In this study, periodic broadcast is implemented in the communications simulator QualNet. Equipped vehicles broadcast to supply their travel time data packets to the neighboring equipped vehicles. In this research, the broadcast interval was set to one second. Each equipped vehicle broadcasts every second, respectively, based on its own release time into a road network, which was randomly distributed. Thus, broadcast times are uniformly distributed over a continuous time interval. This is also how one would want a real system to work, since this minimizes message collisions. An important design consideration for real systems, however, is this timing. If all on-board units were time-synchronized to the GPS clock, and they all chose broadcast times based solely on that clock, then they would all be attempting to communicate at the same time. Even in the real system, therefore, a random stand-off period would need to be built in to spread this demand. Fortunately, many wireless protocols provide for such a mechanism already, including the 802.11 family used in this research.

Travel time data packets are transmitted through a transport layer, a network layer, a MAC (Media Access Control) layer and a physical layer. User Datagram Protocol (UDP) and Internet Protocol (IP) are used for a transport layer and the network layer, respectively. We do not take into account communication routing performing in the network layer since we assume that the information are broadcast to all capable receivers, rather than routed to a specific recipient. For the Medium Access Control (MAC) layer and physical layer, the 802.11a standard is chosen. As mentioned before (subsection 2.13), the Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) scheme makes communication collisions minimized in the

MAC layer. In the physical layer, communication phenomena such as path loss and fading are considered.

5.2.3 Data interpretation

The results of successful communications are sent from QualNet to Paramics every synchronization interval. To synchronize the simulation time between two simulators, we use the Paramics API function "`qpx_NET_timeStep`" called once at the start of each step of simulation time (see code in Appendix A). While users can directly choose a simulation time step between 10 milliseconds and 0.5 seconds in Paramics, we used an effective synchronization interval of one second (because of simulation speed concerns) by invoking this function properly every other 0.5 seconds, and returning without effect on the alternating times. As mentioned in Subsection 4.2.3, the synchronization interval helps to control errors within the model, and the acceptable magnitudes of these errors vary depending on the application.

The results of successful communications obtained from QualNet lead receiving vehicles to update their delivered travel time data packets. The function "transmit" plays the role of updating (see code in Appendix A). Based on the receiver's data set (database), the function "transmit" inserts the sender's travel time data packet into the receiver's data set only if the receiver does not have the data packet. It can be checked with the vehicle ID and the time stamp of data packet whether it exists in the data set. The retention of travel time data for each vehicle is limited to up to 30 of the most recent observations per link. In the preliminary research for individual travel time reliability (Subsection 3.1.3), the number of samples in the worst case

(congested traffic conditions) obtained from the Acceptance Probability was 23 (Figure 3-13), and it was decided to use 30, including 30 % as a safety rate.

The relevance of individual pieces of travel time data on a current traffic condition declines as they move away in time. Depending on how fast this "data relevance" degrades, travel time data packets could become stale and, eventually, useless. According to some staleness threshold set appropriately, old data should be discarded. In the preliminary research for temporal relevance degradation of travel time (Subsection 3.2.3), the correlation coefficients between travel times on many links fell down under 0.5 over 14 minutes (Figure 3-19). Therefore, travel time data packets over 15 minutes old are expired in the current simulation framework. A sender removes data packets more stale than 15 minutes before it transmits. In a real system, these parameters would come from the system design itself or would need to be calibrated to optimize system performance. The point of this study is to showcase the integrated transportation and communications simulation framework, so the parameter choices are not optimized in any systematic way.

5.2.4 Dynamic routing

In a VANET-based traffic information system, equipped vehicles can choose another route to avoid traffic congestion based on the disseminated travel time data. In the simulation framework, the dynamic routing mechanism is the same as it would be in a real system. Whenever equipped vehicles pass split road segments, which means a branch including more than two following links, the Paramics API function "`qpo_RTM_decision`" is called and makes vehicles choose the shortest path (see

code in Appendix A). To find the shortest path, the Dijkstra algorithm (1959) was employed, solving the single-source shortest path problem for a directed graph with non-negative edge weights. The Dijkstra algorithm is used in the function "ShortestPath" (see code in Appendix A). An average of the travel times accumulated from up to 30 records is calculated over each link, and the link distance divided by the speed limit is substituted for links without data in the database. A more robust travel time estimation routine could be used in place of this process, and this is recommended as one of the areas of future research later in this dissertation.

This chapter described the system configuration, map-based travel time data abstraction, and onboard unit configuration for a VANET-based traffic information system. Based on the system configuration mentioned, it was depicted specifically how the simulation framework is built. According to the information system logic, vehicle release, data generation, dissemination, interpretation, and dynamic routing were explained. The next chapter contains a case study based on a real road network. As a result, framework performance, information dissemination speed, and dynamic routing performance are discussed.

Chapter 6: Case Study

This chapter demonstrates an integrated simulation framework of a traffic information system in which vehicles are provided with traffic information through intervehicle communications. Through the case study with a real road network, the framework performance, traffic information dissemination, and dynamic routing performance are discussed on the simulation framework implemented in this research.

6.1 Simulation environment

It is important to attempt to contrive more realistic simulation environments although the purpose of the case study is to assess the simulation framework. A real road network, for example, includes road elements such as road alignment, conflict areas (merging and splitting), and ramps, which could influence traffic movements. A real traffic demand might distribute vehicles throughout the road network.

The road network in this simulation experiment denotes real roadways to compose realistic simulation environment. The roadways include road curvature, merging and splitting areas, and ramps in road sections and interchanges. Figure 6-1 shows the location of the simulation site and the network structure.

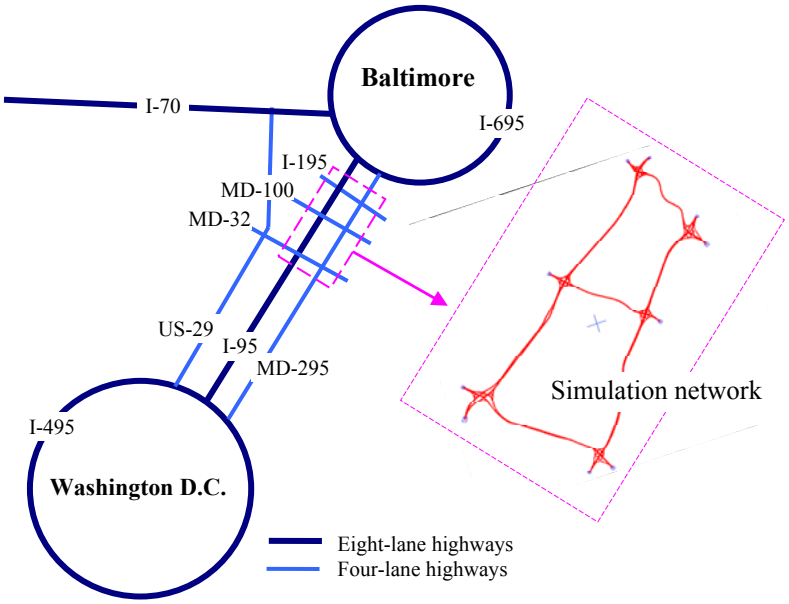


FIGURE 6-1 Simulated road network

The roadway site selected for this simulation is located between Washington, District of Columbia and Baltimore, Maryland in the United States. This highway network (a total of 13 miles, equivalent to 22 km) consists of an eight-lane highway (I-95) and four-lane highways (MD-295, I-195, MD-32, and MD-100), and includes six interchanges. On the termini of the road network, ten traffic demand zones are defined to release vehicles into the network.

It would be necessary that the simulation framework is evaluated in a variety of traffic conditions. Based on 2006 Annual Average Daily Traffic (AADT) data provided by the Maryland State Highway Administration, various traffic demands are established. A half of the AADT on the road that each zone is located is assigned as a traffic demand of the zone since AADT denotes a two-way traffic. We build the ratio table of origin and destination traffics in which the number of vehicles arriving at a

zone is same as the traffic demands released from that zone. Using the assigned traffic demands and the origin-destination ratio table, various traffic demands for traffic conditions from a low density through a high density are generated. Demand Level (DL) is denoted in the range of 1 to 7, DL 1 being one percent of the assigned traffic demands and DL 7 being seven percent of the assigned traffic demands. Figure 6-2 shows the traffic densities during a period of 40 minutes for each DL.

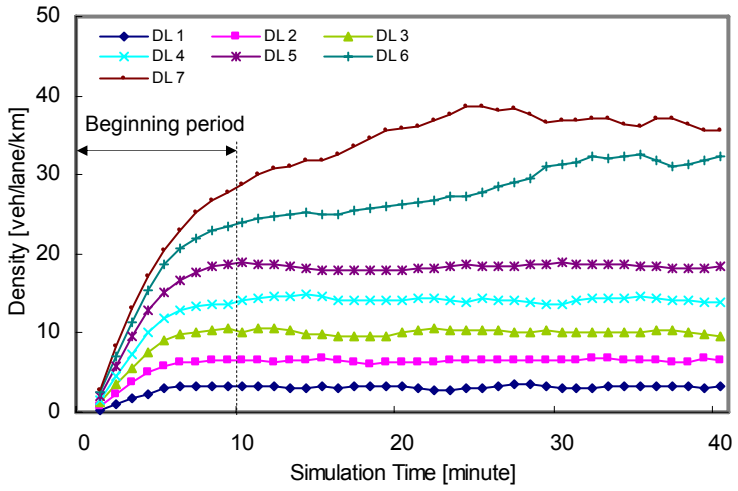


FIGURE 6-2 Traffic demand levels

In Figure 6-2, the traffic density is measured across the whole road network including ramps. The simulation results for the first 10 minutes are excluded since it is regarded as a beginning period. In DLs 1 through 5, the traffic densities are stable after the beginning period, which means any serious congestion does not happen on the road network. The densities in DLs 6 and 7, however, continue to increase because congestions occurred in several interchanges expand. Table 6-1 contains key simulation parameters in experiments.

TABLE 6-1 Simulation parameters

Number of lanes	4 lanes on I-95 and 2 lanes on others per direction
Speed limits	65 mph on I-95 and 55 mph on others
Demand level	DLs 1, 2, 3, 4, 5, 6 and 7
Market penetration	0.5, 1, 3, 5 and 10 [%]
Broadcasting interval	1 second
Protocol	Transport layer: UDP Network layer: IP MAC and Physical layer: 802.11a
Transmission range	250 meters
Simulation time	70 minutes for dynamic routing and 40 minutes for others

In Table 6-1, traffic demands take account of both uncongested conditions and congested conditions. Considering the beginning of the system deployment, this study focuses more on low market penetration. UDP and IP protocols stand for User Datagram Protocol and Internet Protocol, respectively.

For this experiment, a 32-bits personal computer is used (Core 2 Duo processor / 2.4 GHz clock speed, 4 GB memory and Windows XP). Considering high memory usage, 4 GB memory, which is the maximum size of memory in a 32-bits personal computer, is installed. In this computer, Paramics and QualNet run with reciprocal communication via shared memory.

6.2 Framework performance

The VANET simulation is computationally expensive, partly because it needs long computation time and a large amount of memory space since thousands of

vehicles can communicate with each other every ε seconds given. The performance of this simulation framework was measured in terms of ratio of simulation time to computation time (real time). Figures 6-3 and 6-4 show computation time and computer memory usage from 40-minute simulations.

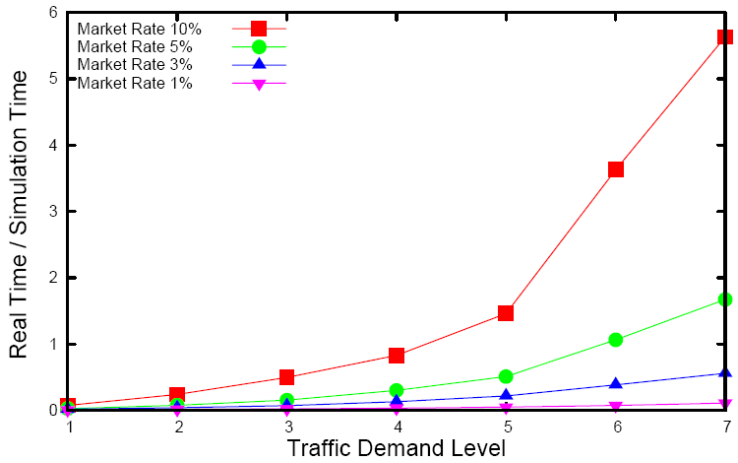


FIGURE 6-3 Computation time

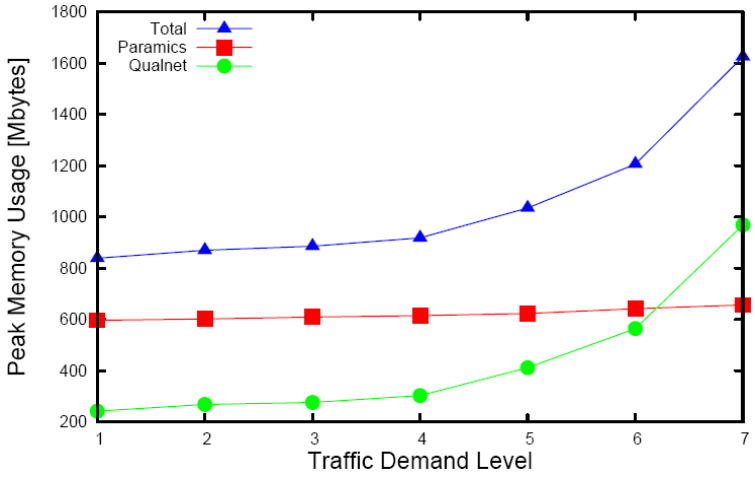


FIGURE 6-4 Computer memory usage

Figure 6-3 illustrates that the simulation time slows exponentially as the traffic becomes heavier. From the demand level 5, simulation works in QualNet certainly increased simulation time. As the number of vehicles grew, the number of communications increased exponentially, which eventually made simulation time slower than real time. The highest ratio of computation time to simulation time was 5.63. In Figure 6-4, the results of memory usage were obtained in 10 % market penetration rate. Memory usage in Paramics did not change very much, whereas that in QualNet significantly became higher from the traffic demand level 5. This reflects that an increase in traffic density considerably boosts communications among equipped vehicles. When approximately 4,000 vehicles (400 equipped vehicles) in the demand level 7 were on the road network, the memory spaces needed for Paramics and QualNet were about 0.7 Gbytes and 1 Gbytes, respectively. Figures 6-5 and 6-6 show the amount of data which Paramics and QualNet exchanged each other for 40-minute simulations.

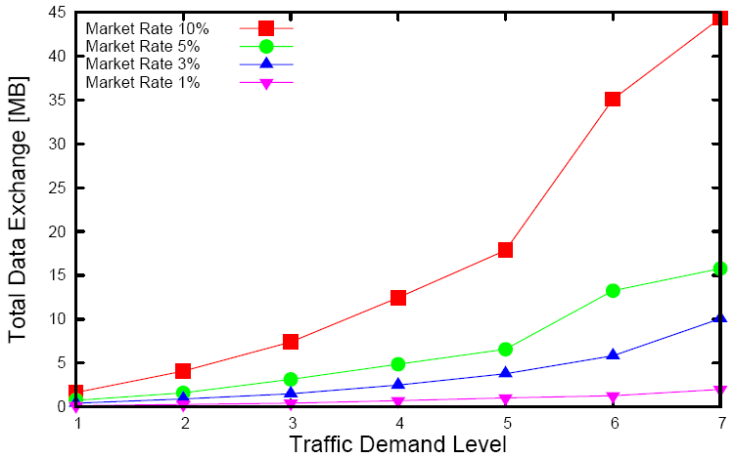


FIGURE 6-5 Total data exchange between simulators

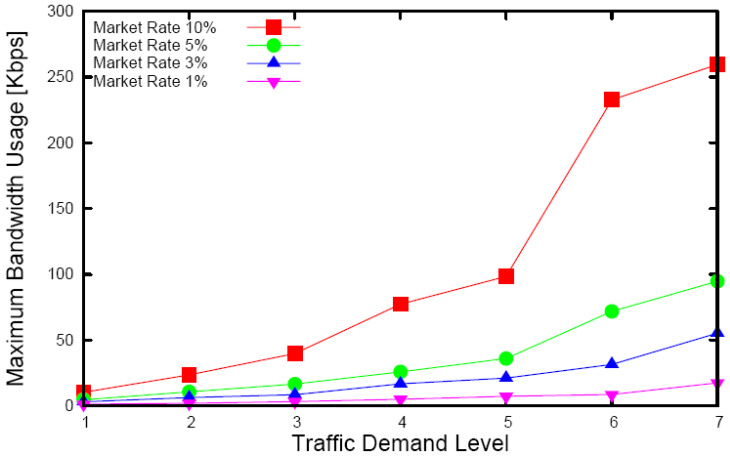


FIGURE 6-6 Maximum data exchange between simulators

Data exchange as well as simulation time and memory usage mentioned above depends on the number of vehicles traveling on the road network. In Figure 6-5, the total amount of exchanged data increases as traffic demand level increases. Although they stiffly increased at the demand level 6, the total of exchanged data increased less at the demand level 7. It appears more conspicuously in Figure 6-6. The maximum amount of the exchanged data increased 140 Kbps more at the demand level 6, whereas it increased only 30 Kbps more at the demand level 7. Figure 6-7 shows broadcast delivery performance from 40-minute simulations.

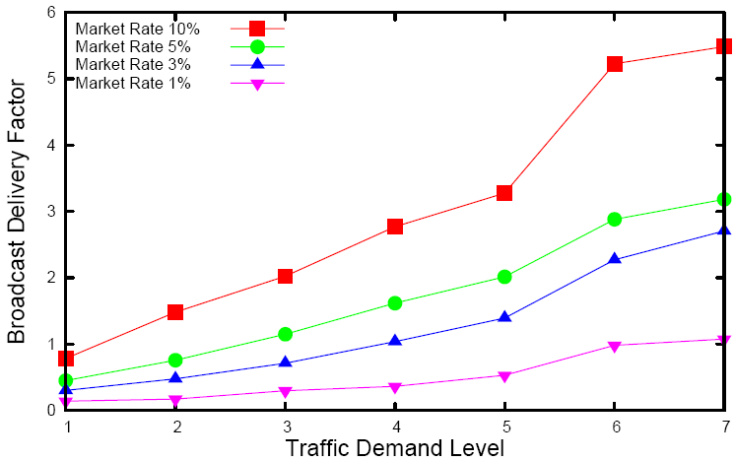


FIGURE 6-7 Broadcast delivery performance

Computation time, memory usage, and data exchange are substantially associated with "broadcast delivery factor" defined as the average number of transferred packets per broadcast from each vehicle. Figure 6-7 shows that the broadcast delivery factor increases as a traffic demand level increases. Compared to a stiff increase at the demand level 6, a broadcast delivery factor slightly increased at the demand level 7. This result explains that more communication collisions occurred in high density traffic condition.

6.3 Traffic information speed

In this section, disseminated speeds of traffic information via broadcast on a real roadway network are investigated in order to evaluate a road network performance. All simulations for traffic information dissemination speed were conducted for 40 minutes. Figure 6-8 show average information speed.

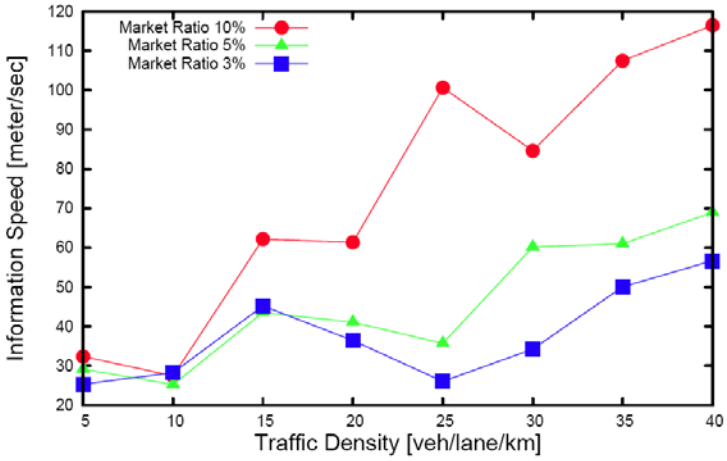


FIGURE 6-8 Average information dissemination

Figure 6-8 was obtained from data throughout the demand levels 1 to 7. Individual travel time data were traced with time and distance from the time when the vehicle released into the network. Using traced data, an average of travel time dissemination speeds was calculated with traffic density every minute. All one-minute speed-density data were aggregated based on density. As shown in Figure 6-8, information speed increases according to density over all market penetration rates. In the low traffic density situations (5 and 10 vehicle/lane.km), information seems to be disseminated via equipped vehicles in the opposite direction; most information speed is around the speed limit (65 mile/hour = 29 meter/sec). In the high traffic density condition (40 vehicle/lane.km), the sufficient availability of equipped vehicles traveling in the same direction reduces the chance to use vehicles in the opposing direction even though it is still possible.

6.4 Dynamic routing performance

As a case study, this simulation framework was applied to investigate the feasibility of dynamic routing mechanism based on traffic information dissemination through inter-vehicular communication. Dynamic routing was conducted using travel time data that were limited up to 30 observations per link as mentioned in Subsection 5.2.3. It determined the shortest path at each split section through Dijkstra's algorithm.

All results were obtained from simulations for 70 minutes. Figures 6-9 and 6-10 show the average travel times of equipped vehicles and unequipped vehicles by market penetration and traffic demand level. For the simulation shown in Figure 6-9, traffic demand level 6 was used, and 10 % market penetration rate was applied in Figure 6-10.

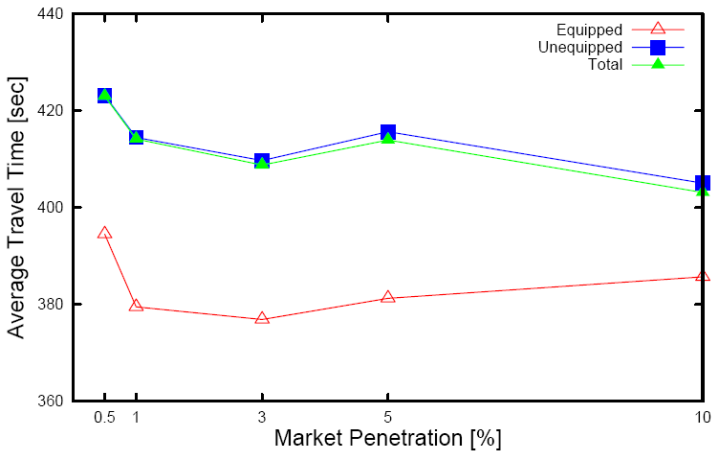


FIGURE 6-9 Dynamic routing performance by market penetration

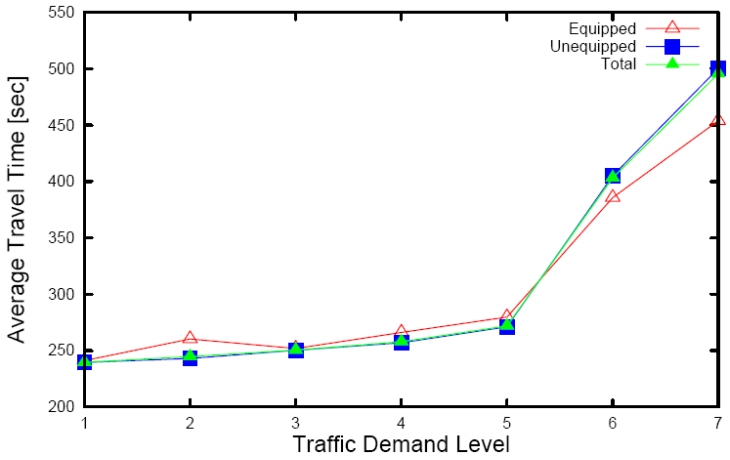


FIGURE 6-10 Dynamic routing performance by traffic demand

As shown in Figure 6-9, it is clear that vehicles equipped with intervehicle communication devices obtain benefits from traffic information dissemination compared to unequipped vehicles. The results show that, even with 0.5 % market penetration rate, the equipped vehicles could gather enough information to avoid traffic congestion. As more equipped vehicles re-route to alternative paths, the overall traffic pattern seems to get better since the average travel time for all the vehicles decreases. However, as the market penetration rate increases, the benefit of re-routing slightly decreases since more vehicles re-route. In this simulation, 3% market penetration rate was the threshold, but the threshold value could change depending on simulation assumptions such as the traffic demand level and the transmission range. As Figure 6-10 indicates, it is clear that re-routing loses its benefits when no congestion is on the road. An unequipped vehicle follows the shortest path based on a link's distance and speed, which means that it always chooses the real shortest path on uncongested condition. The shortest path by limited information could provide an equipped vehicle with a wrong direction.

To evaluate the performance of dynamic routing, a scenario with an incident was simulated. Figure 6-11 shows an incident location and two routes from zone A to zone B, which are the normal route in an uncongested traffic condition and the alternative route in the congested traffic condition caused by the incident.

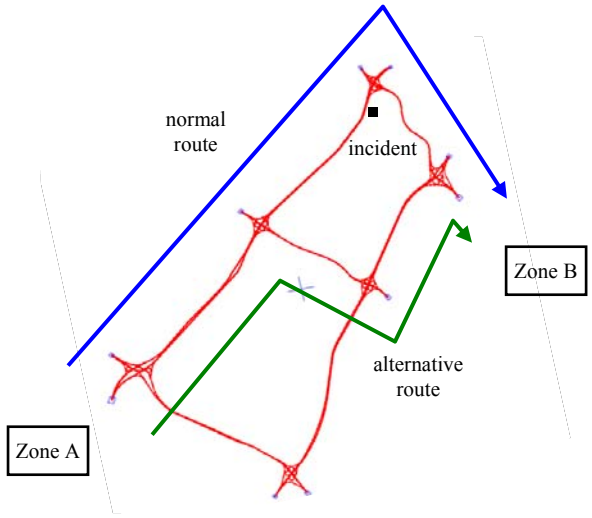


FIGURE 6-11 Incident scenario

In Figure 6-11, the incident which decreases the capacity by 1/3 occurred for 20 minutes (00:30:00 – 00:50:00) during a simulation period of 70 minutes. In an uncongested traffic condition, all vehicles choose the normal route because it is the shortest path. After the congestion caused by this incident happens, equipped vehicles choose the alternative route, whereas equipped vehicles keep following the normal route. Figure 6-12 shows the results obtained at traffic demand level 5, and Figure 6-13 contains the results at market penetration 10 %.

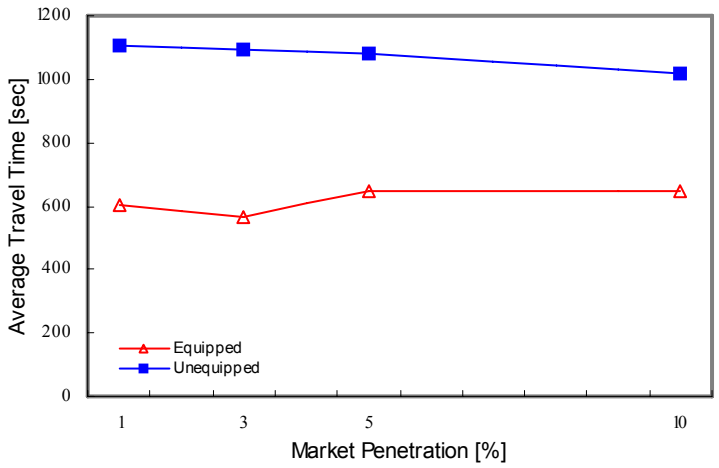


FIGURE 6-12 Dynamic routing performance under incident by market penetration

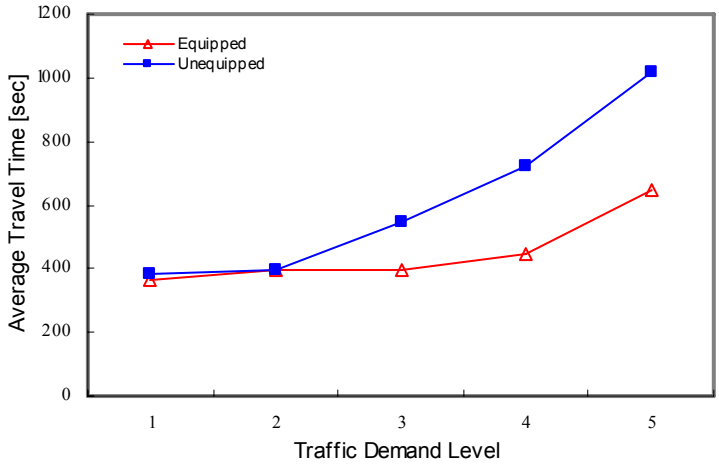


FIGURE 6-13 Dynamic routing performance under incident by traffic demand

Both Figure 6-12 and Figure 6-13 show conspicuous benefits by dynamic routing. In Figure 6-12, the incident in this scenario caused the traffic condition heavily congested. As a result, the difference in travel time between equipped and unequipped vehicles became large. In Figure 6-13, while the incident did not influence the traffic conditions at demand levels 1 and 2, it increased the difference in

travel times from demand level 3. In fact, it is not important how many seconds this traffic information system allow equipped vehicles to save in this incident scenario since it depends on simulation parameters such as an incident location and incident time periods. It is, however, important that the equipped vehicles could gather enough information to avoid traffic congestion even in the low market penetration.

6.5 Discussion

In this chapter, the simulation framework from this dissertation was used to develop experimental cases. These experiments deployed a VANET-based traffic information system to evaluate the performance of the framework. They were conducted based on a real road network and real traffic demands, and the results demonstrated that this system is capable of providing traffic information even in a low market penetration. In particular, equipped vehicles conspicuously obtained benefits to save travel time in comparison with unequipped ones. The road network used in this research was the simple road network, which provides fewer opportunities to turn to an alternative route. Clearer results might be expected on a road network with more alternative routes. The next chapter reviews the entire works in this dissertation, and shows the contributions to this research.

Chapter 7: Conclusion

The ongoing efforts to apply advanced technologies to help solve transportation problems advanced the growing trend of integrating mobile wireless communications into transportation systems. In particular, VANETs based on ad hoc networks allow vehicles to constitute a decentralized information dissemination system on roadways and to share their own information.

This dissertation presented some work on information dissemination and spatial and temporal degradation of information. This is an important issue to understand in the decentralized, autonomous information system likely to prevail with VANETs. The first results of the dissertation were developed in a relatively simple simulation framework, partly to highlight the fact that some general results are possible without sophisticated tools, but also to show the envelope where this argument loses strength. The dissertation also included the development of a conjoined transportation and communication simulation framework to evaluate the decentralized system based on a VANET, and showed its implementation on a traffic information system. This chapter summarizes the whole research effort and describes the interpretation of the results obtained from the experiments. The dissertation is concluded with a summary of the contributions this work has made to the body of research on VANETs.

7.1 Summary of Findings

As part of this research, we developed an integrated simulation framework for VANET applications in which the characteristics of transportation and wireless

communications were embedded. For practical vehicle movements such as car following, lane changing, and shock waves, a transportation simulator (Paramics) was employed. A communication simulator (QualNet) was chosen for wireless communications characteristics such as path loss, fading, interference, and communication collision. For the implementation of this framework, these simulators were tightly coupled and finely synchronized in terms of simulation time and node (vehicle) mobility, facilitated by their respective APIs.

The implemented simulation framework was evaluated on a traffic information system with various traffic demands and market penetration rates based on a real road network. For framework performance, simulation time (Figure 6-3), memory usage (Figure 6-4), data exchange (Figures 6-5 and 6-6), and the number of delivered nodes (Figure 6-7) were investigated. While these performance metrics degraded gradually in uncongested traffic conditions, they changed much more precipitously in congested traffic conditions. The slopes of the data exchange and the number of delivered nodes metrics were, however, less severe in a jammed traffic condition. Since the metrics depend on vehicle density, normally, an increase in traffic density induces an exponential increase in the communications among vehicles. Nevertheless, some metrics show a lower slope in high density conditions, due to the fact that the actual number of successful communications is reduced beyond a certain density due to an increase in message collisions. Fortunately, with robust protocols, message collisions do not take as much time as successful messages to resolve; hence communications systems tend to treat these congested communications conditions rather gracefully.

For traffic information system performance, information in the low traffic density situations (5 and 10 vehicle/lane.km) seems to be delivered primarily via equipped vehicles traveling in the opposite direction, given that most of the recorded information speeds are less than the speed limit (65 mile/hour = 29 meter/sec). In the high traffic density condition (40 vehicle/lane.km), the average of information speed (117 meter/second) in the 10 % market penetration rate scenario seems to be reasonable compared to the maximum transmission speed (250 meter/second). Based on these results, traffic information speed in a VANET is sufficiently fast to deliver reliable information in low density conditions as well as high density conditions. Dynamic routing conducted based on delivered traffic information was effected in congested traffic conditions rather than uncongested ones, as would be expected.

7.2 Contribution

This dissertation treated research issues on inter-vehicle communications for transportation applications. With the spread of wireless communications devices, many research studies have been conducted for a variety of transportation applications under the topic of VANETs. While the computer simulation approach is a popular evaluation method in this field, previous researches have not provided simulation frameworks fully satisfied both in the transportation and communications domains. By developing an integrated transportation and communication simulation framework for VANET applications, this dissertation has contributed to the research on VANETs as follows.

- State-of-the-art research related to VANETs was reviewed. In particular, the critical limitations of previous simulation framework results were disclosed.
- Basic studies on information value and the degradation of that value were offered. These studies offer insights into the ways that these decentralized and autonomous data sources can provide inputs into algorithms that differ from how current versions of these algorithms – fed from fixed sensors at known locations – might operate.
- The system model that was designed through this research can include most applications in VANETs. It is expected to be used as a base to develop applications for VANETs.
- This research implemented a VANET-based information model into an integrated transportation and communication simulation framework in which these independent simulation tools were tightly coupled and finely synchronized.
- A traffic information system as a VANET application was built based on the simulation framework developed in this research. In this system, vehicles record their own travel time data, share these data via an ad hoc network, and reroute at split sections based on stored travel time data. The programming code used to build this application is attached in Appendix. It is expected to be used for experiments to simulate various traffic situations in a VANET.
- The sensitivity for simulation loads was shown as a function of traffic demands and market penetration. It is expected to help design a simulation framework.

- Information speeds on a real roadway network were obtained. In this research, information speeds were approximately between the road speed limit - in which case they were mostly delivered by vehicles traveling on the opposite direction - and half of the transmission range (250/2 meter) per second, which means they were delivered by vehicles traveling in the same direction.
- Successful dynamic routing based on stored traffic data was demonstrated in this framework. The benefits from dynamic routing were shown, which previous studies have not shown.

This chapter described the findings obtained through the entirety of this dissertation, and summarized the contributions to the research on VANETs, particularly simulation work. This research focused on the development of an integrated transportation and communication simulation framework to build a more realistic environment with which to study VANETs, as compared to previous studies. It is believed that a wide range of VANET applications can be designed and assessed using methodologies influenced by and contributed to by the simulation framework and other methods developed in this dissertation.

Appendix A

Paramics API code

```
#define _CRT_SECURE_NO_DEPRECATE 0
#define REMOTE_HOST "10.0.0.2"
#define REMOTE_PORT 2000
#define REALLOC_FACTOR 10
#define SKIP_SEC 0

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <float.h>
#include <winsock2.h>
#include <assert.h>
#include "programmer.h"
#include "hash.h"
//#include "queue.h"

#define MAXLINK 686
#define MAX_VCNT 3000
#define MAX_RCNT 2000
#define MAX_NODES 500
#define MAX_Q2P_PKT_SIZE  (12+MAX_VCNT*(8+MAX_RCNT*16))
#define SHMBUF_SIZE 2000*1000*16*4 //204800000

int MarketRate = 50;          // Market penetration rate [0.1%]
int shortest_path = 1; // 1 yes, 0 no

float CommInterval = 3;       // transmission interval of equipped vehicles [sec]
float CommRange = 100.0; // [meter]
int NumOfLinks;           // # of links
int NumOfNodes;
int MaxTT = 30;           // Maximum number of Travel time structures
int ExpiryTime = 1800;        // Expired time difference from current time [sec]
int MaxGTT = 4000;            // Maximum # of Global Travel time structures 6000veh/4lane.hr

float QualnetTime = 0; //NEW ALGORITHM

// Global Travel Time structure by linked list
struct TravelTime_s
{    int VHCID;    //4
     int LinkIndex; //2 byte
     float TravelTime;//4
     float ExitTime;//
     float x; //infospeed
     float y; //infospeed
};
```

```
// Vehicle Travel Time structure by linked list
struct VHC_TravelTime_s
{
     float updated_time;
     struct TravelTime_s* GTT;
};

struct VHC_TTDB_s
{
   struct VHC_TravelTime_s VHC_TT[MAXLINK][30];
};

// Vehicle user data structure
typedef struct VHC_USERDATA_s
{    int VHCID;
     Bool equipped;              // Bool = int, yes: 1, no: 0
     float ReleaseTime;
     float EntryTime;
     struct VHC_TTDB_s *db;
     int  DataSize;
} VHC_USERDATA;

FILE *outResult, *TTResult, *densityResult, *errout;

int sockfd;

int *GTTIndex; ;      // Index array for Global TT array
struct TravelTime_s** GTT;       // Global TT structure pointer array

float TotalMile;

// --- Prototypes ----------
void transmit(VEHICLE* source, VEHICLE* target, float updated_time);
int CompareGTT(const void* x, const void* y);
void PrintStatistics();
void got_new_data( struct VHC_TravelTime_s *TT); //infospeed
// --- End of Prototypes ---

// --- qualnet update begins ---

struct QU_trsmt  // vehicle receiving packet
{
     int svid;
     int rvid;
     float rtime;
};

typedef struct
{
     int total_size;
     float qualnet_time;
     int trsmt_cnt;
     struct QU_trsmt *trsmt;
}QualnetUpdate;

QualnetUpdate qu;
```

```
float ParseQualnetUpdate(char *buff, int size);

int LinkMap[MAXLINK];
void InitLinkMap();
int GetLinkMap(int link);

// shared memory
typedef struct {
    HANDLE shmHandle;
    HANDLE shmMutex;
    LPCTSTR shmBuf;
    int shmSize;
}SHMComm;

SHMComm shmemPQ, shmemQP;

SHMComm SHMCommConnect(char *shmName, int bufSize);
SHMComm SHMCommCreate(char *shmName, int bufSize);
int SHMCommWrite(SHMComm shmComm, char *buf, int bufSize);
int SHMCommRead(SHMComm shmComm, char *buf);
void SHMCommClose(SHMComm shmComm);

int SOCKCommConnect(char *hostname, int port);
int SOCKCommCreate(char *hostname, int port);
int SOCKCommWrite(int fd, char *buf, int bufSize);
int SOCKCommRead(int fd, char *buf);
void SOCKCommClose(int fd);

int ShortestPath(VHC_USERDATA* vudata, int stt, int end);

// we try to avoid calling realloc every time with these
// pre-allocated global communication buffers
char *buffPQ;    // communication buffer  P->Q
char *buffQP;   // communication buffer  Q->P
int buffPQSize; // buffer size
int buffQPSize;

typedef struct p2q_vehicle {
    int        vid;
    float     x;
    float     y;
    int        pkt_size;
} P2QVehicle;

typedef struct p2q_packet_s {
    float     time;
    int  vcnt;
    P2QVehicle      vhcl[MAX_VCNT];
} P2QPacket;

P2QPacket pkt;

typedef struct all_vehicle_s VehicleLnk;

struct all_vehicle_s {
```

```
    VEHICLE *v;
    VHC_USERDATA *vudata;
    int vid;
    VehicleLnk *next;
};

VehicleLnk *empty_lnk;
VehicleLnk *vehicle_lnk;
int vehicle_cnt;
VehicleLnk gvlink_vehicles[MAX_VCNT];

WBHASH *hash;

unsigned int HashFunct(void *nullitem, void *item)
{
   unsigned int val = 0;
   VehicleLnk *pv = (VehicleLnk*) item;

   nullitem = nullitem;

     val = pv->vid;

   WBTrcReturn(WBTRC_HASH,val,("%d",val));
}

int Compare(void *nullitem, char *item1, char *item2)
{
    VehicleLnk *pv1=(VehicleLnk*) item1, *pv2=(VehicleLnk*) item2;
    nullitem = nullitem;

    if(pv1->vid < pv2->vid) return 1;
    else if(pv1->vid > pv2->vid) return -1;
    else return 0;
}

void qpg_VHC_hash_init()
{
    if ((hash = WBHashOpen(NULL,50)) != NULL){
        WBHashHashingF(hash,NULL,HashFunct);
        WBHashCompareF(hash, NULL, (int (*) ()) Compare);
        //WBHashExecuteF(hash, NULL, (int (*) ()) Execute);
    }
}

void qpg_VHC_hash_add(VehicleLnk *pv)
{
    WBHashAdd(hash, pv);
}

VehicleLnk* qpg_VHC_hash_lookup(int vid)
{
    VehicleLnk pv;
    pv.vid = vid;
    return WBHashSearch(hash, &pv);
}
```

```
void qpg_VHC_hash_remove(VehicleLnk *pv)
{
     WBHashDelete(hash, pv);
}

void qpg_VHC_hash_close()
{
    WBHashClose(hash);
}

int LinkMap[MAXLINK];
int Junction[MAXLINK];
int NodeMap[MAX_NODES];
float *SPTT[MAX_NODES];

void InitMap()
{
     FILE *fp;
     char line[1024];
     char tmp1[256], tmp2[256];
     int node1, node2;
     char *token;
     int start_node, end_node;
     int i;
     int first_link;
     LINK *link;

     // node map
     for(i=0;i<MAX_NODES;i++)
          NodeMap[i]=i;

     fp = fopen("links", "rt");
     if(fp == NULL) {
          fprintf(errout, "no links file\n");
          return;
     }

     while(!feof(fp)){
          fgets(line, 1024, fp);

          sscanf(line, "%s %s", tmp1, tmp2);
          if(!strcmp(tmp1, "link")){
               node1 = atoi(tmp2);
          }
          if(!strcmp(tmp1, "on-ramp")){
               fgets(line, 1024, fp);
               sscanf(line, "%s %s %d", tmp1, tmp2, &node2);
               NodeMap[node2] = node1;

          }
     }

     fclose(fp);

     // link map
     for(i=0;i<MAXLINK;i++){
```

```
    LinkMap[i]=i;
}

fp = fopen("MergedLinks.txt", "rt");
if(fp == NULL) return;

while(!feof(fp)){
    fgets(line, 1024, fp);

    first_link = -1;

    token = strtok(line, "-");
    end_node = atoi(token);
    while(token)
    {
        start_node = end_node;
        end_node = atoi(token);

        for ( i = 1 ; i <= NumOfLinks ; i++)
        {
                link = qpg_NET_linkByIndex(i);
                if( start_node != atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)))) continue;
                else if( end_node != atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)))) continue;
                else {
                    if(first_link < 0) first_link = i;
                    LinkMap[i] = first_link;
                    break;
                }

        }
        token = strtok(NULL, "-");

    };
}
fclose(fp);

for ( i = 0 ; i <= NumOfLinks ; i++) Junction[i] = 0;

fp = fopen("junctions", "rt");
if(fp == NULL) {
    fprintf(errout," junctions doesn't exist\n");
    return;
}

while(!feof(fp)){
    fgets(line, 1024, fp);

    sscanf(line, "%s %s", tmp1, tmp2);

    if(!strcmp(tmp1, "junction")){
        token = strtok(tmp2, ":");
        start_node = atoi(token);
        token = strtok(NULL, " ");
        end_node = atoi(token);

        for ( i = 1 ; i <= NumOfLinks ; i++)
```

```
            {
                link = qpg_NET_linkByIndex(i);
                if( start_node != atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)))) continue;
                else if( end_node != atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)))) continue;
                else {
                    Junction[i] = 1;
                    break;
                    }

            }
        }

    }

    fclose(fp);
}


gvlink_initialize() {
    VehicleLnk *p;
    empty_lnk = gvlink_vehicles;
    for( p = gvlink_vehicles; p < gvlink_vehicles+MAX_VCNT-1; p++)
        p->next = p+1;
    p->next = NULL;

    vehicle_lnk = NULL;
    vehicle_cnt = 0;

    // hash init
    qpg_VHC_hash_init();
}

VehicleLnk *gvlink_first_vehicle() { return vehicle_lnk; }

void gvlink_print(char *s)
{
    VehicleLnk *p;
    VHC_USERDATA *vudata;

    return;

    fprintf(errout, "vehicles: ");
    fprintf(errout, s );

    for( p = vehicle_lnk; p; p = p->next )
    {
        vudata = p->vudata;

        if(vudata)
            fprintf(errout, "(%p:%d:%d:%d:%.2f:%.2f)->", p->v, p->vid, qpg_VHC_uniqueID(p->v), vudata-
>equipped,
                vudata->ReleaseTime, vudata->EntryTime );
        else
            fprintf(errout, "(%p:%d:%d)->", p->v, p->vid, qpg_VHC_uniqueID(p->v) );
    }
    fprintf(errout, "\n");
```

```
    fflush(errout);
}

gvlink_add_vehicle( VEHICLE *v, VHC_USERDATA* data)
{
    VehicleLnk *nv;

    if( ! empty_lnk )
    {
        fprintf(errout, "\nno more memory for vehicle" );
        exit(1);
    }
    nv = empty_lnk;
    empty_lnk = empty_lnk->next;


    // safety code
    if(!qpg_VHC_uniqueID(v))
        v = qpg_VHC_original(v);

    nv->v = v;
    nv->next = vehicle_lnk;
    nv->vudata = data;
    vehicle_lnk = nv;
    vehicle_cnt++;

    // hash add
    nv->vid = qpg_VHC_uniqueID(v);

    qpg_VHC_hash_add(nv);

}

struct VHC_USERDATA_s* gvlink_get_userdata( VEHICLE *v)
{
    VehicleLnk *p;

    int vid;

    // for safety
    if(!qpg_VHC_uniqueID(v))
        v = qpg_VHC_original(v);

    for( p = vehicle_lnk; p; p = p->next )
    {
        vid = qpg_VHC_uniqueID(v);
        if(p->vid == vid){
            return p->vudata;
        }
    }

    return NULL;
}

VehicleLnk* gvlink_get_vehicle(int vid)
{
```

```
    VehicleLnk *p;

    for( p = vehicle_lnk; p; p = p->next )
    {
        if(p->vid == vid){
            if(qpg_VHC_uniqueID(p->v) != p->vid){
                fprintf(errout, "p->vid (%d) != qpg_VHC_uniqueID(%d)\n", p->vid, qpg_VHC_uniqueID(p->v));
                fflush(errout);
                continue;
            }
            return p;
        }
    }

    return NULL;
}

void gvlink_delete_vehicle(int vid)
{
    VehicleLnk *p, *q, *found, *prev;

    if( ! vehicle_lnk )
    {
        fprintf(errout, "\nno more vehicle to delete" );
        exit(1);
    }

    // safety code

    if( qpg_VHC_uniqueID(vehicle_lnk->v) == vid )
    {
        prev = NULL;
        found = vehicle_lnk;
    }
    else
    {
        int cnt=0;
        for( p = vehicle_lnk, q = vehicle_lnk->next; q && q->vid != vid; p = q, q = q->next )
        {
            ;
        }

        if( !q )
        {
            fprintf(errout, "\nno such vehicle %d found to delete", vid );
            fflush(errout);
            return;
        }
        prev = p;
        found = q;
    }

    if( !prev )
        vehicle_lnk = found->next;
    else
        prev->next = found->next;
```

```
    if(found->vudata->db)
        free(found->vudata->db);
    if(found->vudata)
        free(found->vudata);

    found->next = empty_lnk;
    empty_lnk = found;
    vehicle_cnt--;

    // hash delete
    found->vid = qpg_VHC_uniqueID(found->v);
    qpg_VHC_hash_remove(found);
    found->vid = -1;

}


// ----------------------------------------------------------------------
// called once when the full network has been read into modeller
// ----------------------------------------------------------------------
void qpx_NET_postOpen(void)
{
    int i;
    LINK* link;

    char outputname[256];

    errout = fopen("erroutput.txt", "w");

    if(shortest_path)
        sprintf(outputname,"report-%dsec-mr%.1f-shortest.txt", qpg_CFG_duration(), (float)MarketRate/10.);
    else
        sprintf(outputname,"report-%dsec-mr%.1f.txt", qpg_CFG_duration(), (float)MarketRate/10.);

    outResult = fopen(outputname, "w");
    fprintf(outResult, "Duration Time = %d\n", qpg_CFG_duration());
    fprintf(outResult, "Market Rate = %.1f\n", (float)MarketRate/10.);
    if(shortest_path) fprintf(outResult, "ShortestPath\n");
    else fprintf(outResult, "NO ShortestPath\n");

    if(shortest_path)
        sprintf(outputname,"TTreport-%dsec-mr%.1f-shortest.txt", qpg_CFG_duration(), (float)MarketRate/10.);
    else
        sprintf(outputname,"TTreport-%dsec-mr%.1f.txt", qpg_CFG_duration(), (float)MarketRate/10.);

    TTResult = fopen(outputname, "w");
    fprintf(TTResult, "Duration Time = %d\n", qpg_CFG_duration());
    fprintf(TTResult, "Market Rate = %.1f\n", (float)MarketRate/10.);
    if(shortest_path) fprintf(outResult, "ShortestPath\n");
    else fprintf(outResult, "NO ShortestPath\n");

    if(shortest_path)
        sprintf(outputname,"density-%dsec-mr%.1f-shortest.txt", qpg_CFG_duration(), (float)MarketRate/10.);
    else
        sprintf(outputname,"density-%dsec-mr%.1f.txt", qpg_CFG_duration(), (float)MarketRate/10.);
```

```
    densityResult = fopen(outputname, "w");
    fprintf(densityResult, "Duration Time = %d\n", qpg_CFG_duration());
    fprintf(densityResult, "Market Rate = %.1f\n", (float)MarketRate/10.);
    if(shortest_path) fprintf(densityResult, "ShortestPath\n");
    else fprintf(densityResult, "NO ShortestPath\n");

// initialize random number generator seed
    srand((unsigned) time(NULL));

    NumOfLinks = qpg_NET_links();
    NumOfNodes = qpg_NET_nodes();

   qps_GUI_printf(" --- Paramics Programmer API: Vehicular Ad hoc Network --- \n");
   qps_GUI_printf(" --- Number of Links: %d --- \n", NumOfLinks);

    fprintf(errout, " --- Paramics Programmer API: Vehicular Ad hoc Network --- \n");
   fprintf(errout, " --- Number of Links: %d --- \n", NumOfLinks);
    fflush(errout);

    GTTIndex = (int*) malloc((NumOfLinks+1)*sizeof(int));
    memset( GTTIndex, 0, (NumOfLinks+1)*sizeof(int));

    GTT = (struct TravelTime_s**) malloc((NumOfLinks+1)*sizeof(struct TravelTime_s*));
    if(GTT == NULL){
        fprintf(errout, "malloc error: %d\n", __LINE__);
        fflush(errout);
    }
    for( i = 0 ; i < (NumOfLinks+1) ; i++ )     { // Allocate GTT memory
        GTT[i] = (struct TravelTime_s *) malloc(MaxGTT*sizeof(struct TravelTime_s));

        if(GTT[i]==NULL){
            fprintf(errout, "malloc error: %d\n", __LINE__);
            fflush(errout);
        }
    }

    for( i =0; i<MAX_NODES; i++)
        SPTT[i] = (float*) malloc(MAX_NODES*sizeof(float));

    InitMap();

    // qualnet update initialization
    qu.trsmt = malloc(MAX_VCNT*MAX_RCNT*sizeof(struct QU_trsmt));
    if(qu.trsmt==NULL) {
        fprintf(errout, "QualnetUpdate: malloc error\n");
        fflush(errout);
        exit(1);
    }

    gvlink_initialize();

#ifdef HIGHWAY_SHMEMLIB

    shmemPQ = SHMCommCreate("P2Q", SHMBUF_SIZE);
    shmemQP = SHMCommCreate("Q2P", SHMBUF_SIZE);
```

```
    fprintf(errout, "\nSHMCommCreate done.\n");
    fflush(errout);

#elif HIGHWAY_SOCKET

    sockfd = SOCKCommConnect(REMOTE_HOST, REMOTE_PORT);

    fprintf(errout, "\nSocket connect success: %s\n", REMOTE_HOST);
    fflush(errout);
#endif

    buffPQ = malloc(MAX_Q2P_PKT_SIZE );
    buffQP = malloc(MAX_Q2P_PKT_SIZE );

    buffPQSize = MAX_Q2P_PKT_SIZE ;
    buffQPSize = MAX_Q2P_PKT_SIZE ;

    //density

    TotalMile = .0;
    for(i=0; i<MAXLINK; i++){

        link = qpg_NET_linkByIndex(i);
        if(link == NULL) continue;
        if(qpg_LNK_barred(link)) continue;

        TotalMile += (float) qpg_LNK_lanes(link) * qpg_LNK_length(link) / 1609.0;  // mile
    }
    fprintf(densityResult, "Total Mileage = %f\n", TotalMile);
    fflush(densityResult);
}

Bool qpo_RTM_enable(void)
{
    if(shortest_path) return TRUE;
    else return FALSE;
}

#if 1
int qpo_RTM_decision(LINK *link, VEHICLE *vehicle)
{
    int i;
    int prevpos, curpos;
    int end_candidate[256];
    int next_node;
    int nextlink_end;
    LINK *nextlink;
    ZONE* zone;
    VHC_USERDATA* vudata;

    int nlinks;
    int nexitlinks;

    int link_index = qpg_LNK_index(link);
```

```
    if(shortest_path == 0) return 0;

    if(Junction[link_index] == 0) return 0;

    // safety code
    if(!qpg_VHC_uniqueID(vehicle))
        vehicle = qpg_VHC_original(vehicle);

    vudata = gvlink_get_userdata(vehicle);

    if(!vudata) return 0;

    if(!vudata->equipped) {
        return 0;
    }

    // this is not equipped vehicle
    vudata = gvlink_get_userdata(vehicle);

    // nonvalid user data
    if(vudata == NULL){
        fprintf(errout, "warning: vudata is null. vid = %d, RTM_decision is called before the car is released.\n",
qpg_VHC_uniqueID(vehicle));
        fflush(errout);
        return 0;
    }

    prevpos = atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)));
    curpos = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)));

    zone = qpg_NET_zone(qpg_VHC_destination(vehicle));

    nlinks = qpg_ZNE_links(zone);

    for(i=1;i<=nlinks;i++){
        end_candidate[i] = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(qpg_ZNE_link(zone, i))));
    }

    nexitlinks = qpg_LNK_exitLinks(link);

    if(nexitlinks<2) return 0;

    for(i=1;i<=nlinks;i++) {
        next_node = ShortestPath(vudata, curpos, end_candidate[i]);

        if(next_node != 0){
            next_node = NodeMap[next_node];
            break;
        }
    }

    if(prevpos == 358 && curpos == 4 && next_node == 170)
        return 0;

    if(prevpos == 298 && curpos == 269 && next_node == 294)
        return 0;
```

```
    for(i=1; i<= nexitlinks ; i++){
        nextlink = qpg_LNK_exit(link, i);

        nextlink_end = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(nextlink)));
        nextlink_end = NodeMap[nextlink_end];

        if( next_node == nextlink_end){
            return i;
        }
    }

    fprintf(errout,"check this out\n");

    return 0;
}
#endif

#if 0

void  qpo_RTM_nextLink(LINK* link, VEHICLE* vehicle, int nextout, LINK* *nextlink, int *newdestp)
{
    int i;
    int prevpos, curpos;
    int end_candidate[256];
    int next_node;
    int candidatelink_end;
    LINK* candidatelink;
    ZONE* zone;
    VHC_USERDATA* vudata;

    int nlinks;
    int nexitlinks;

    int link_index = qpg_LNK_index(link);

    fprintf(errout," nextout = %d\n", nextout);
    fflush(errout);

    *nextlink = qpg_LNK_exit(link, nextout+1);

    // safety code
    if(!qpg_VHC_uniqueID(vehicle))
        vehicle = qpg_VHC_original(vehicle);

    vudata = gvlink_get_userdata(vehicle);
    if(vudata == NULL) return;

    if(!vudata->equipped) return;

    if(shortest_path == 0) return;

    if(Junction[link_index] == 0) return;

    // nonvalid user data
    if(vudata == NULL){
```

```
        fprintf(errout, "vudata is null (%d), vid= %d, exitTime= %d \n", __LINE__, qpg_VHC_uniqueID(vehicle),
qpg_VHC_existTime(vehicle));
        fprintf(errout, "!!! original pointer. vid = %d\n", qpg_VHC_uniqueID(qpg_VHC_original(vehicle)));
        fflush(errout);

        return;
    }

    prevpos = atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)));
    curpos = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)));

    zone = qpg_NET_zone(qpg_VHC_destination(vehicle));

    nlinks = qpg_ZNE_links(zone);

    for(i=1;i<=nlinks;i++){
        end_candidate[i] = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(qpg_ZNE_link(zone, i))));
    }

    nexitlinks = qpg_LNK_exitLinks(link);

    if(nexitlinks>1 && vudata->equipped )
    {
        for(i=1;i<=nlinks;i++) {
            next_node = ShortestPath(vudata, curpos, end_candidate[i]);

            if(next_node != 0){
                next_node = NodeMap[next_node];
                break;
            }
        }

        for(i=1; i<= nexitlinks ; i++){
            candidatelink = qpg_LNK_exit(link, i);
            if(candidatelink == NULL){
                fprintf(errout, "candidatelink is null %d\n", __LINE__);
                fflush(errout);
            }

            candidatelink_end = atoi(qpg_NDE_name(qpg_LNK_nodeEnd(candidatelink)));
            candidatelink_end = NodeMap[candidatelink_end];

            if( next_node == candidatelink_end){

                if(nextout != i){
                }

                *nextlink = candidatelink;
                return;
            }
        }

        fprintf(errout,"check this out\n");

        *nextlink = qpg_LNK_exit(link, nextout+1);
        return;
```

```
    }

    *nextlink = qpg_LNK_exit(link, nextout+1);
    return;

}
#endif


// ----------------------------------------------------------------------
// called once at the start of each time step of simulation time.
// ----------------------------------------------------------------------
void  qpx_NET_timeStep(void)
{
    static int qualnet_running = TRUE;

    float CurrentTime = qpg_CFG_simulationTime();
    int nrecv;

    VHC_USERDATA *vudata;
    struct VHC_TravelTime_s* vttlink = NULL;
    float z, b, g;
    int cnt;
    VehicleLnk *pv;

    static stop = 0;
    static time_t elapsed=0;
    clock_t st, et;

    static float accumulated_density=0;
    static int accumulated_step=0;

    if( (int)(CurrentTime*10) % 10 == 5) return;

    if(CurrentTime < SKIP_SEC) return;

    if(CurrentTime > (float) qpg_CFG_duration() - 6.0 && stop == 0) {
        PrintStatistics();
        stop = 1;
    }

    st = clock();

    pkt.time = CurrentTime;

    pv = gvlink_first_vehicle();
    cnt = 0;

    while(pv)
    {
        VehicleLnk *to_delete;

        if( qpg_LNK_index(qpg_VHC_link(pv->v)) < 0 )
        {
            to_delete = pv;
            pv = pv->next;
```

```
            gvlink_delete_vehicle(to_delete->vid);

            continue;
        }

        vudata = pv->vudata;
        if(vudata == NULL){
            fprintf(errout, ">>>>>>>>vudata == null\n");
            fflush(errout);

            to_delete = pv;
            pv = pv->next;
            gvlink_delete_vehicle(to_delete->vid);
            continue;
        }

        if(pv->vid != vudata->VHCID){
            gvlink_print("invalid_vhcid"); //debug
            fprintf(errout, ">>>>>>>>>vudata->VHCID (%d) != pv->vid (%d) real:%d\n", vudata->VHCID, pv-
>vid,qpg_VHC_uniqueID(pv->v));
            fflush(errout);

            to_delete = pv;
            pv = pv->next;
            gvlink_delete_vehicle(to_delete->vid);

            continue;
        }

        pkt.vhcl[cnt].vid = pv->vid;
        qpg_POS_vehicle(pv->v, qpg_VHC_link(pv->v), &pkt.vhcl[cnt].x, &pkt.vhcl[cnt].y, &z, &b, &g );

        pkt.vhcl[cnt].pkt_size = vudata->DataSize * 12; /*sizeof(struct TravelTime_s)*/;

        if(pkt.vhcl[cnt].pkt_size < 0) {
            fprintf(errout, "pkt.vhcl[%d].pkt_size = %d\n", cnt, pkt.vhcl[cnt].pkt_size);
            fflush(errout);

            to_delete = pv;
            pv = pv->next;

            gvlink_delete_vehicle(to_delete->vid);

            pv = pv->next;
            continue;
        }

        if(pkt.vhcl[cnt].pkt_size > 655355){
            fprintf(errout, "pkt.vhcl[%d].pkt_size = %d\n", cnt, pkt.vhcl[cnt].pkt_size);
            fflush(errout);
            pkt.vhcl[cnt].pkt_size = 655355;
        }

        pv = pv->next;
        cnt++;
```

```
    }

    pkt.vcnt = cnt;

    accumulated_density += qpg_NET_vehiclesSimulating()/TotalMile;
    accumulated_step++;
    fprintf(densityResult, "Time %f Density= %f  EquippedVCNT= %d   TotalVCNT= %d\n", CurrentTime,
qpg_NET_vehiclesSimulating()/TotalMile, pkt.vcnt, qpg_NET_vehiclesSimulating());
    fflush(densityResult);

    if( ((int)(CurrentTime*10)/10) % 60 == 0){
        fprintf(densityResult, "[***AVG***] Time %f Density= %f \n", CurrentTime,
accumulated_density/accumulated_step);
        fflush(densityResult);
        accumulated_density = .0;
        accumulated_step = 0;
    }


#ifdef HIGHWAY_SHMEMLIB

    if(MAX_Q2P_PKT_SIZE  < sizeof(int)+sizeof(float)+pkt.vcnt*sizeof(P2QVehicle) )
        fprintf(errout, "packet size is too big..[%d]\n", sizeof(int)+sizeof(float)+pkt.vcnt*sizeof(P2QVehicle));

    if( SHMCommWrite( shmemPQ, (char *)&pkt, sizeof(int)+sizeof(float)+pkt.vcnt*sizeof(P2QVehicle) ) < 0 )
    {
        fprintf(errout, "\nWrite error\n");
        fflush(errout);
        fclose(errout);
        exit(1);
    }
#elif HIGHWAY_SOCKET
    if( SOCKCommWrite(sockfd, (char *)&pkt, sizeof(int)+sizeof(float)+pkt.vcnt*sizeof(P2QVehicle) )  < 0 )
    {
        fprintf(errout, "\nWrite error\n");
        fflush(errout);
        fclose(errout);
        exit(1);
    }
#endif

    // check if qualnet is running
    if( qualnet_running )
    {

#if defined(HIGHWAY_SHMEMLIB) || defined(HIGHWAY_SOCKET)
    do {

#ifdef HIGHWAY_SHMEMLIB

        nrecv = SHMCommRead(shmemQP, buffQP);

#elif HIGHWAY_SOCKET

        nrecv = SOCKCommRead(sockfd, buffQP);
#endif
```

```
            if ( nrecv < 0 ) {
                fprintf(errout, "\nCommRead() Error" );
                fprintf(errout, "\nCurrentTime = %.10f\n", CurrentTime);
                fflush(errout);
                break;
            }
            else if (nrecv > 0 )
            {
                QualnetTime = ParseQualnetUpdate(buffQP, nrecv);

                if( QualnetTime < 0 )
                {
                    qualnet_running = FALSE;
                }
            }

        } while( qualnet_running && QualnetTime < CurrentTime );

#endif

        }
        else{
            qpx_NET_close();
            exit(0); // if qualnet is running
        }

        et = clock();
        elapsed += et - st;
}


// ----------------------------------------------------------------------
// As each vehicle is released into the network create a new lookup
// record for it.
// ----------------------------------------------------------------------
void qpx_VHC_release(VEHICLE* vehicle)
{
    VHC_USERDATA *data;

    int i,j;
    int vid;

    // check for a bad vehicle
    if(!vehicle)
        return;

    // safety code
    if(qpg_VHC_uniqueID(vehicle))
        vehicle = qpg_VHC_original(vehicle);

    vid = qpg_VHC_uniqueID(vehicle);

    data = calloc(1, sizeof(VHC_USERDATA));
```

```
    data->VHCID = qpg_VHC_uniqueID(vehicle);
    data->ReleaseTime = qpg_CFG_simulationTime();
    data->EntryTime = -1;
    data->DataSize = 0; // data size

    if ( MarketRate >= (rand() % 1000) + 1)
    {
        data->equipped = TRUE;

        data->db = calloc(1, sizeof(struct VHC_TTDB_s));
        for(i=0;i<MAXLINK;i++) {
            for(j=0; j<MaxTT; j++){
                data->db->VHC_TT[i][j].GTT = NULL;
                data->db->VHC_TT[i][j].updated_time=0.0;
            }
        }

        gvlink_add_vehicle(vehicle, data);

    }
    else
    {
        data->equipped = FALSE;
        data->db = NULL;

        qps_VHC_userdata(vehicle, data);
    }

}


// ------------------------------------------------------------------------
// store travel time into USERDATA structure whenever vehicles
// pass nodes
// ------------------------------------------------------------------------
void qpx_VHC_transfer(VEHICLE* vehicle, LINK* link1, LINK* link2)
{
    VHC_USERDATA *vudata;

    int i, j;
    int tt_cnt;
    float z;
    float len, limit;

    // safety code
    if(qpg_VHC_uniqueID(vehicle))
        vehicle = qpg_VHC_original(vehicle);

    vudata = gvlink_get_userdata(vehicle);
    if(!vudata) return;

    if(vudata->db == NULL) return;

    // if entry is -1, this is first transfer and we don't make TT data
    if( vudata->EntryTime < 0 )
    {
```

```
        vudata->EntryTime = qpg_CFG_simulationTime();
        return;
    }

    // store travel time in Global travel time linked list
    if( LinkMap[qpg_LNK_index(link1)] == LinkMap[qpg_LNK_index(link2)]) return;

    i = LinkMap[qpg_LNK_index(link1)];
    j = GTTIndex[i];
    GTT[i][j].VHCID = vudata->VHCID;
    GTT[i][j].LinkIndex = LinkMap[qpg_LNK_index(link1)];
    GTT[i][j].ExitTime = qpg_CFG_simulationTime();
    GTT[i][j].TravelTime = GTT[i][j].ExitTime - vudata->EntryTime;

    len = qpg_LNK_length(link1);
    limit = qpg_LNK_speedlimit(link1);

    qpg_POS_node(qpg_LNK_nodeEnd(link1), &GTT[i][j].x, &GTT[i][j].y, &z ); //infospeed
    GTTIndex[i]++;
    if (GTTIndex[i] == MaxGTT){
        fprintf(errout,"GTTIndex == MaxGTT\n");
        GTTIndex[i] = 0;
    }

    vudata->EntryTime = GTT[i][j].ExitTime;

    // store travel time in travel time linked list

    for(tt_cnt=0; tt_cnt<MaxTT; tt_cnt++) {
        if(vudata->db->VHC_TT[i][tt_cnt].GTT == NULL )
            break;
    }

        vudata->db->VHC_TT[i][tt_cnt].GTT = NULL;
        vudata->DataSize++;
    }
    else{
        // replacement
        vudata->db->VHC_TT[i][MaxTT-1].GTT= &GTT[i][j];
        vudata->db->VHC_TT[i][MaxTT-1].updated_time = GTT[i][j].ExitTime;
    }

    qsort(vudata->db->VHC_TT[i], tt_cnt, sizeof(struct VHC_TravelTime_s), CompareGTT);
}


void qpx_VHC_arrive(VEHICLE* vehicle, LINK* link, ZONE* zone)
{
    VHC_USERDATA *data;

    int dest, org;

    float CurrentTime = qpg_CFG_simulationTime();

    // safety code
    if(qpg_VHC_uniqueID(vehicle)==0)
```

```
        vehicle = qpg_VHC_original(vehicle);

    data = gvlink_get_userdata(vehicle);
    if(!data) data = qpg_VHC_userdata(vehicle);

    dest = qpg_ZNE_index(qpg_NET_zone(qpg_VHC_destination(vehicle)));
    org = qpg_ZNE_index(qpg_NET_zone(qpg_VHC_origin(vehicle)));

    // equipped vehicle
    if(data->equipped)
    {
        if (org == 5 && dest == 10)
        {
            fprintf(TTResult,"\n( %d -> %d ) \t rel: %f \t tt: %f equipped vhcid: %d", org, dest, data->ReleaseTime,
qpg_CFG_simulationTime()-data->ReleaseTime, data->VHCID);
            fflush(TTResult);
        }

        // free TT DB
        gvlink_delete_vehicle(qpg_VHC_uniqueID(vehicle));

    }
    // nonequipped vehicle
    else {
        fprintf(TTResult,"\n( %d -> %d ) \t rel: %f \t tt: %f unequipped vhcid: %d", org, dest, data->ReleaseTime,
qpg_CFG_simulationTime()-data->ReleaseTime, data->VHCID);
        fflush(TTResult);

        free(data);
    }

}


// ---------------------------------------------------------------------
// called once when the full network has been closed into modeller
// ---------------------------------------------------------------------
void qpx_NET_close(void)
{
    int i;

    for( i = 0 ; i < (NumOfLinks+1) ; i++ )     // Allocate GTT memory
        free(GTT[i]);

    free(GTT);
    free(GTTIndex);

    for( i =0; i<MAX_NODES; i++)
        free(SPTT[i]);

    // qualnet update finalization
    free(qu.trsmt);

    free(buffPQ);
    free(buffQP);
```

```
#ifdef HIGHWAY_SHMEMLIB

    SHMCommClose(shmemPQ);
    SHMCommClose(shmemQP);

#elif HIGHWAY_SOCKET

    SOCKCommClose(sockfd);

#endif

    fprintf(errout, "\n qpx_NET_close is called \n");

    fclose(errout);
    fclose(outResult);
    fclose(TTResult);
    fclose(densityResult);

}

VEHICLE *global_current_vehicle;
LINK *global_current_link;


//---------------------------------------------------------------------
// Function name: transmit
// Parameters:    VEHICLE *source: host vehicle
//                VEHICLE *target: guest vehicle
// Return value:  void
// Description:   Transmit travel time data of host vehicle to guest vehicle
//---------------------------------------------------------------------
void transmit(VEHICLE* source, VEHICLE* target, float updated_time)
{
    int i,j;
    int tp, sp;
    int copied;
    int cnt_t;

    struct VHC_TravelTime_s c[30];
    struct VHC_TravelTime_s *s, *t;

    VHC_USERDATA *sourcedata = gvlink_get_userdata(source);      // host
    VHC_USERDATA *targetdata = gvlink_get_userdata(target); // guest

    if(sourcedata == NULL){
        fprintf(errout, "sourcedata is null , vid = %d\n", qpg_VHC_uniqueID(source));
        fflush(errout);
        return;
    }
    if(targetdata == NULL){
        fprintf(errout, "targetdata is null, vid = %d\n", qpg_VHC_uniqueID(target));
        fflush(errout);

        gvlink_get_userdata(target);
        return;
    }
```

```
        global_current_vehicle = target;
        global_current_link = qpg_VHC_link(target);

    for(i=0;i<MAXLINK;i++){
        s = sourcedata->db->VHC_TT[i];
        t = targetdata->db->VHC_TT[i];

        copied = 0;
        tp=0; sp=0;
        for(j=0;j<30;j++){
            if( t[tp].GTT == s[sp].GTT){
                c[j] = t[tp++];
                sp++;
            }
            else if( CompareGTT(&t[tp], &s[sp]) < 0 ) {
                c[j] = t[tp];
                tp++;
            }
            else if( CompareGTT(&t[tp], &s[sp]) > 0 ) {
                c[j] = s[sp];
                c[j].updated_time = updated_time;

                copied++;
                sp++;
            }
            else {
                fprintf(errout,"this shouldn't happen. check this out!, line %d\n", __LINE__);
                fflush(errout);
            }
            if(c[j].GTT == NULL) break;
        }

        if(copied>0){
            for(j=0;j<30;j++){
                if(t[j].GTT==NULL) break;
            }
            cnt_t = j;

            for(j=0;j<30;j++){
                if(c[j].GTT==NULL) break;
            }

            if( cnt_t != j) {
                assert( j > cnt_t);
                targetdata->DataSize += (j-cnt_t);;
            }

            memcpy(t, c, MaxTT*sizeof(struct VHC_TravelTime_s));
        }

    }

    return;
}
```

```
// -------------------------- Shared Mem Communication API DEFINITIONS -------------------------//

// create shared memory region
// returns when it accepts a connection
SHMComm SHMCommCreate(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    shmComm.shmHandle = CreateFileMapping(
        INVALID_HANDLE_VALUE,    // use paging file
        NULL,                    // default security
        PAGE_READWRITE,          // read/write access
        0,                       // max. object size
        bufSize,                 // buffer size
        shmName);

    if (shmComm.shmHandle == NULL)
    {
        fprintf(errout, "Could not create file mapping object (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
          FILE_MAP_ALL_ACCESS, // read/write permission
          0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout,"Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    memset(shmComm.shmBuf, 0, bufSize);

    *((int *)shmComm.shmBuf) = 0;
    *((int *)(shmComm.shmBuf + sizeof(int))) = 0;

    sprintf(name_buf, "%sMutex", shmName);
    // create mutex
    shmComm.shmMutex = CreateMutex(
        NULL,  // default security attributes
        FALSE,  // initially not owned
        name_buf);

    if (shmComm.shmMutex == NULL)
    {
        fprintf(errout,"Could not create mutex lock (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    shmComm.shmSize = bufSize;
```

```
    return shmComm;

}


SHMComm SHMCommConnect(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    // !!! infinite loop to open shared memory
    while( NULL == (shmComm.shmHandle = OpenFileMapping(
        FILE_MAP_ALL_ACCESS,   // read/write access
        FALSE,                 // do not inherit the name
        shmName))) Sleep(1000);

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
          FILE_MAP_ALL_ACCESS, // read/write permission
          0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout, "Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    sprintf(name_buf, "%sMutex", shmName);
    // !!! another infinite loop to open mutex
    while ( NULL == (shmComm.shmMutex = OpenMutex(
        MUTEX_ALL_ACCESS,      // request full access
        FALSE,                 // handle not inheritable
        name_buf))) Sleep(1000);

    shmComm.shmSize = bufSize;

    return shmComm;
}

int SHMCommWrite(SHMComm shmComm, char *buf, int _size) {
    DWORD waitResult;
    int head, rear;  // head, rear of circular queue
    char *cq;          // circular queue
    int cq_size; // circular queue size
    int first_half, second_half;
    int data_size, block_size;

    if(NULL == buf) {
        fprintf(errout, "ERROR - SHMCommWrite: buf is NULL..\n");
        fflush(errout);
        return -1;
    }

    // if data size is not divided by 4, we append some nulls
    if(0 != _size%4) {
        data_size = _size + (4 - _size%4);
        fprintf(errout, "WARNING - SHMCommWrite: size is not divided by 4. \n");
```

```
            fflush(errout);
        }
        else
            data_size = _size;

        block_size = data_size + sizeof(int);

    LABEL:
        while(1){
            waitResult = WaitForSingleObject(
                shmComm.shmMutex,   // handle to mutex
                5000L);   // five-second time-out interval
            if(waitResult == WAIT_OBJECT_0)
                break; // got mutex lock
        }

        // now mutual exclusion block starts from here

        // first 4 byte points to the head of circular queue
        // second 4 byte points to the rear of circular queue where new data should be appended
        head = *((int *)shmComm.shmBuf);
        rear = *((int *)(shmComm.shmBuf + sizeof(int)));
        cq = (char*) (shmComm.shmBuf + 2*sizeof(int));
        cq_size = shmComm.shmSize - 2*sizeof(int);

        if(rear < head) {
            // check whether new rear would exceed head
            if( (rear+block_size)>=head ) {
                fprintf(errout, "ERROR - SHMCommWrite: Shared memory buffer is filled up.\n");
                fflush(errout);

                ReleaseMutex(shmComm.shmMutex);
                exit(1);
            }

            memcpy(cq+rear, &data_size, sizeof(int));
            memcpy(cq+rear+sizeof(int), buf, _size);

            rear = (rear + block_size) % cq_size;
        }
        else if( (rear+block_size) > cq_size ){
            // need to wrap around
            // check whether new rear would exceed head
            if( (rear + block_size - cq_size) >= head ) {
                fprintf(errout, "ERROR - SHMCommWrite: Shared memory buffer is filled up.\n");
                fflush(errout);
                ReleaseMutex(shmComm.shmMutex);
                goto LABEL;
            }

            memcpy(cq+rear, &data_size, sizeof(int));

            first_half = cq_size - (rear + sizeof(int));
            second_half = _size - first_half;

            memcpy(cq+rear+sizeof(int), buf, first_half);
```

```
            memcpy(cq, buf + first_half, second_half);

            rear = ( rear + block_size ) % cq_size;
        }
        else {
            // nothing to worry about
            memcpy(cq+rear, &data_size, sizeof(int));
            memcpy(cq+rear+sizeof(int), buf, _size);
            rear = ( rear + block_size ) % cq_size;

        }

        // mutual exclusion block ends here

        if (! ReleaseMutex(shmComm.shmMutex)) {
            fprintf(errout, "\n0) Error for release (%d)\n", GetLastError() );
            fflush(errout);
            return -1;
        }

        return block_size;
}

int SHMCommRead(SHMComm shmComm, char *buf) {
        DWORD waitResult;
        int head, rear;  // head, rear of circular queue
        char *cq;          // circular queue
        int cq_size; // circular queue size
        int first_half, second_half;
        int size;

        if(NULL == buf) {
            fprintf(errout, "ERROR - SHMCommRead: buf is NULL..\n");
            fflush(errout);
            return -1;
        }

        while(1){
            waitResult = WaitForSingleObject(
                shmComm.shmMutex,   // handle to mutex
                5000L);   // five-second time-out interval
            if(waitResult== WAIT_OBJECT_0)
                break; // got mutex lock
        }
        // now mutual exclusion block starts from here
        // first 4 byte points to the head of circular queue
        // second 4 byte points to the rear of circular queue where new data should be appended
        head = *((int *)shmComm.shmBuf);
        rear = *((int *)(shmComm.shmBuf + sizeof(int)));
        cq = (char*) (shmComm.shmBuf + 2*sizeof(int));
        cq_size = shmComm.shmSize - 2*sizeof(int);
        if(head == rear) {
            if (! ReleaseMutex(shmComm.shmMutex)) {
                fprintf(errout, "\n2) Error for release (%d)\n", GetLastError() );
                fflush(errout);
                return -1;
```

```
            }
            // no data ready
            return 0;
        }


        size = *((int *)(cq+head));

        if(size > MAX_Q2P_PKT_SIZE ) {
            fprintf(errout, "WARNING!!!: packet is too big :%d bytes\n", __FILE__, __LINE__, size);
            fprintf(errout, "INCREASE MAX_Q2P_PKT_SIZE . \n", size);
            fflush(errout);
        }

        if( (head + (int)sizeof(int) + size) > cq_size){
            // wrap around
            first_half = cq_size - head - sizeof(int);
            second_half = size - first_half;

            memcpy(buf, cq + head + sizeof(int), first_half);
            memcpy(buf+first_half, cq, second_half);

            head = ( head + size + sizeof(int) )% cq_size;
        }
        else {

            // nothing to worry about
            memcpy(buf, cq+head+sizeof(int), size);
            head = ( head + size + sizeof(int) )% cq_size;
        }

        *((int *)shmComm.shmBuf) = head;

        // mutual exclusion block ends here
        if (! ReleaseMutex(shmComm.shmMutex)) {
            fprintf(errout, "\n1) Error for release (%d)\n", GetLastError() );
            fflush(errout);
            return -1;
        }
        return size;
    }


    void SHMCommClose(SHMComm shmComm){
        // unmap shaered memory
        UnmapViewOfFile(shmComm.shmBuf);

        CloseHandle(shmComm.shmHandle);
        // end of shared memory
    }


    int SOCKCommCreate(char *hostname, int port)
    {
        int fd=0;
        // Qualnet creates socket..
```

```
    return fd;
}

int SOCKCommConnect(char *hostname, int port)
{
    int fd;
    struct hostent *server;
    struct sockaddr_in servAddr;
    int errcode;
    u_long arg = 1; int err; // non-blocking socket

    // socket open
    if ((fd = socket(AF_INET, SOCK_STREAM, 0/*IPPROTO_TCP*/)) < 0){
            fprintf(errout, "\nSocket open error\n");
            fflush(errout);
    }

    //setsockopt(sd,SOL_SOCKET,SO_SNDBUF,&soptval,sizeof (soptval));

    server = gethostbyname(hostname);

    memset ( (char*) &servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(port);
  memcpy( (char *)&servAddr.sin_addr.s_addr, (char *)server->h_addr, server->h_length);

    while ( connect(fd, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0) {
        errcode = WSAGetLastError();
        fprintf(errout, "\nSocket connect error: %s, %d\n", hostname, errcode);
        fflush(errout);
    }

    err = ioctlsocket(fd, FIONBIO, &arg);
  if (err)
  {
     fprintf(errout, "Error setting socket to non-blocking mode, err = \"%s\"",
                WSAGetLastError());
        fflush(errout);
        assert(1);
  }

    return fd;
}


int SOCKCommWrite(int sock, char *buf, int size)
{
    int nsend;
    int remaining_size;
    char *remaining_data;

    remaining_size = size;
    remaining_data = buf;

    while( remaining_size > 0 )
    {
```

```
        if( (nsend = send(sock, (const char*) remaining_data, remaining_size, 0)) < 0 ) {
            if( nsend == SOCKET_ERROR ){
                int err = WSAGetLastError();
                if( err == WSAEWOULDBLOCK ) // no data ready for read
                    continue;
                fprintf(errout, "\nERROR: SOCKCommWrite()\n");
                fflush(errout);
                exit(1);
            }
        }
        if( nsend <= remaining_size )
        {
            remaining_size -= nsend;
            remaining_data += nsend;
        }

    }

    return size;
}

int SOCKCommRead(int fd, char* buf)
{
    int nrecv;
    int size, remaining_size;
    char* remaining_data = buf;


    // this will take care of the size of qualnet update packet
    nrecv = recv(fd, (char*) &size, sizeof(int), 0);
    if(nrecv<0) return 0;

    fprintf(errout, "data to be read : %dbyte\n", size);
    fflush(errout);

    *(int*)buf = size;
    remaining_size = size - nrecv;
    remaining_data += nrecv;

    while(remaining_size){
        nrecv = recv(fd, (char*) remaining_data, remaining_size, 0);

        if( nrecv == 0 ) continue;
        if( nrecv == SOCKET_ERROR )
        {
            int err = WSAGetLastError();
            if( err == WSAEWOULDBLOCK ) // no data ready for read
                continue;
            else if( err == WSAECONNRESET ) // connection closed
            {
                fprintf(errout, "Socket closed by qualnet\n");
                fflush(errout);
                fclose(errout);
                exit(1);
            }
            else if( err == WSAEMSGSIZE )
```

```
            {
                fprintf(errout, "Data too big for buffer" );
                fflush(errout);
                fclose(errout);
                exit(1);
            }
        }

        if( nrecv <= remaining_size ){
            remaining_size -= nrecv;
            remaining_data += nrecv;
        }
    }

    return size;
}


void SOCKCommClose(int fd)
{
    closesocket(fd);
}


int CompareQUrtime(const void* x, const void* y)
{
    // low to high
    struct QU_trsmt *a= (struct QU_trsmt*) x;
    struct QU_trsmt *b= (struct QU_trsmt*) y;

    if(a->rtime - b->rtime > 0) return 1;
    else if(a->rtime - b->rtime < 0) return -1;
    else return 0;
}

int CompareGTT(struct VHC_TravelTime_s *a, struct VHC_TravelTime_s *b)
{
    // high to low

    if(a->GTT==NULL && b->GTT==NULL) return 0;
    if(a->GTT==NULL) return 1;
    if(b->GTT==NULL) return -1;

    if(a->GTT->ExitTime > b->GTT->ExitTime) return -1;
    else if(a->GTT->ExitTime < b->GTT->ExitTime) return 1;
    else {
        if(a->GTT->VHCID > b->GTT->VHCID) return -1;
        else if(a->GTT->VHCID < b->GTT->VHCID) return 1;
        else return 0;
    }
}


//---------------------------------------------------------------------
// Update TT according to communication results provided from Qualnet
//---------------------------------------------------------------------
```

```
float ParseQualnetUpdate(char *buff, int size)
{
    VehicleLnk *source, *target;

    int i,j;
    int ptr=0;
    int rvhc_cnt;
    int svhc_cnt;
    int rvid;

    qu.total_size = *(int*)buff;
    ptr += sizeof(int);

    qu.qualnet_time = *(float*)(buff+ptr);

    ptr += sizeof(float);

    rvhc_cnt = *(int*)(buff+ptr);
    ptr += sizeof(int);

    if(rvhc_cnt > MAX_VCNT){
        fprintf(errout, "ERROR - ParseQualnetUpdate: qu.rvhc_cnt > MAX_VCNT\n");
        exit(1);
    }

    qu.trsmt_cnt=0;
    for(i=0; i<rvhc_cnt; i++)
    {
        rvid = *(int*)(buff+ptr);
        ptr += sizeof(int);

        svhc_cnt = *(int*)(buff+ptr);
        ptr += sizeof(int);

        if(svhc_cnt > MAX_RCNT){
            fprintf(errout, "ERROR - ParseQualnetUpdate: svhc_cnt [%d] > MAX_RCNT\n", svhc_cnt);
            exit(1);
        }

        for(j=0; j<svhc_cnt; j++){
            qu.trsmt[qu.trsmt_cnt].rvid = rvid;

            qu.trsmt[qu.trsmt_cnt].rtime = *(float*)(buff+ptr);   // update time
            ptr += sizeof(float);

            qu.trsmt_cnt++;
        }
    }

    qsort(qu.trsmt, qu.trsmt_cnt, sizeof(struct QU_trsmt), CompareQUrtime);

    for(i=0; i<qu.trsmt_cnt; i++)
    {
        source = gvlink_get_vehicle(qu.trsmt[i].svid);
        if( !source )
        {
```

```
            fprintf(errout, "ParseQualnetUpdate: src vid %d do not exist\n", qu.trsmt[i].svid );
            fflush(errout);
            gvlink_delete_vehicle(qu.trsmt[i].svid);
        }
        target = gvlink_get_vehicle(qu.trsmt[i].rvid);
        if( !target )
        {
            fprintf(errout, "ParseQualnetUpdate: dst vid %d do not exist\n", qu.trsmt[i].rvid);
            fflush(errout);
            gvlink_delete_vehicle(qu.trsmt[i].rvid);
        }

        if( source && target ) {
            transmit(source->v, target->v, qu.trsmt[i].rtime);
        }
    }

    return qu.qualnet_time;
}


int ShortestPath(VHC_USERDATA* vudata, int stt, int end)
{
    int previous[MAX_NODES];// previous node
    int v[MAX_NODES];           // Permanent label array (1: permanent, 0: undefined)
    float ttarr[MAX_NODES];         // Travel time array from stt
    float min;                  // Temporary smallest cost

    int i, j, k;
    LINK *link, *nextlink;
    int nextlinks;
    int n1, n2, n3, n4;

    float tm;
    int cnt;

    float link_len, speed;
    float speed_limit;
    float angle1, angle2;
    int need_split;

    if(!vudata) return 0;

    // initialize with a large number
    for ( i = 0 ; i < MAX_NODES ; i++ ){
        for( j = 0 ; j < MAX_NODES ; j++) {
            if(i!=j) SPTT[i][j] = FLT_MAX;
            else SPTT[i][j] = .0;
        }
    }

    // adjust with optimal time
    for ( k = 1 ; k < MAXLINK ; k++)
    {
        link = qpg_NET_linkByIndex(k);
        if(link == NULL) continue;
```

```
    if(qpg_LNK_barred(link)) continue;

    i = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)))];    // start node
    j = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)))];      // end node

    if(qpg_LNK_speedlimit(link) != 0){
        link_len = qpg_LNK_length(link);    // meter
        speed_limit = qpg_LNK_speedlimit(link);
        speed = 1000 * speed_limit* 1.609;   // meter/hr
        SPTT[i][j] = 3600 * link_len / speed; // 3600 * hour
    }
    else {
        fprintf(errout,"qpg_LNK_speedlimit(link) is 0.. OTL\n");
        fflush(errout);
    }
}

// update with what this vehicle knows of
for(i=0; i<MAXLINK; i++)
{
    tm=.0; cnt=0;

    for(j=0; j<MaxTT; j++)
    {
        tm += vudata->db->VHC_TT[i][j].GTT->TravelTime;
        cnt++;
    }

    if(cnt>0 && tm!=0) {
        link = qpg_NET_linkByIndex(i); //vulink->LinkIndex);
        n1 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)))];    // start node
        n2 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)))];      // end node

        link_len = qpg_LNK_length(link);
        speed_limit = qpg_LNK_speedlimit(link);

        SPTT[n1][n2] = tm/cnt; // average travel time for this link
    }
}

// safety code. handle junctions!!!

for(i=0; i<MAXLINK; i++)
{
    link = qpg_NET_linkByIndex(i);
    if(link == NULL) continue;
    if(qpg_LNK_barred(link)) continue;

    n1 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeStart(link)))];
    n2 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeEnd(link)))];

    angle1 = 2.0*3.14*qpg_LNK_endAngle(link)/360.0;

    need_split = 0;

    nextlinks = qpg_LNK_exitLinks(link);
```

```
    if(nextlinks>1){
        for(j=1; j<=nextlinks; j++)
        {
            nextlink = qpg_LNK_exit(link, j);

            angle2 = 2.0*3.14*qpg_LNK_endAngle(nextlink)/360.0;

            n3 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeStart(nextlink)))];
            n4 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeEnd(nextlink)))];

            if( cos(angle1)*cos(angle2)+sin(angle1)*sin(angle2) < 0 ||
                (n1 == 358 && n3 == 4 && n4 == 170) ){

                if(n1 == 162 && n3 == 4 && n4 == 170)
                    continue;

                need_split = 1;
                break;
            }
        }

        if(need_split){
            for(j=1; j<=nextlinks; j++)
            {
                nextlink = qpg_LNK_exit(link, j);
                angle2 = 2.0*3.14*qpg_LNK_endAngle(nextlink)/360.0;

                n3 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeStart(nextlink)))];
                n4 = NodeMap[atoi(qpg_NDE_name(qpg_LNK_nodeEnd(nextlink)))];

                if( cos(angle1)*cos(angle2)+sin(angle1)*sin(angle2) > 0 ||
                    (n1 == 162 && n3 == 4 && n4 == 170) ){

                    if(n1 == 358 && n3 == 4 && n4 == 170)
                    continue;

                    assert(n2 == n3);

                    SPTT[n1][n4] = SPTT[n1][n2] + SPTT[n3][n4];

                }
            }
            SPTT[n1][n2] = FLT_MAX;
        }
    }
}


for( i = 0 ; i < MAX_NODES ; i++ )   // initialize
{   v[i] = 0;           // undefined
    ttarr[i] = FLT_MAX;      // infinite
    previous[i] = INT_MAX;      // undefined
}

ttarr[stt] = 0;                  // set the cost of the start node
```

```
    // iterate as such the number of nodes
    for( i = 1, k = INT_MAX ; i < MAX_NODES ; i++ )
    {
        // Set the currently minimum cost
        for( j = 1, min = FLT_MAX ; j < MAX_NODES ; j++ )
            if(( v[j] == 0 ) && ( ttarr[j] < min ))
            {
                k = j;
                min = ttarr[j];
            }

        if ( k == end )          // reach destination
            break;

        v[k] = 1;                // set a permanent label

        if( min == FLT_MAX )
            break;

        // Calculate the smallest cost
        for( j = 1 ; j < MAX_NODES ; j++ )
            if (( stt != j ) && ( ttarr[j] > ttarr[k] + SPTT[k][j]))
            {
                ttarr[j] = ttarr[k] + SPTT[k][j];
                previous[j] = k;
            }

    }


    // Find the next node of the start node
    i = end;
    while( previous[i] != stt ){
        if(end == 359) {
        }
        i = previous[i];

        if(i==INT_MAX) {
            return 0;
        }
    }
    return i; // i: next node
}


void PrintStatistics()
{
     int i,j;
     float total_time=.0, total_avg=.0, temp=.0, avg_lane=.0;
        float* vehicle_tt;
        int max_vcnt=80000;

        vehicle_tt = (float*) malloc(max_vcnt*sizeof(float));
        if(vehicle_tt == NULL) fprintf(errout, "malloc error %d\n", __LINE__);
```

```
        for(i=0;i<max_vcnt;i++)
            vehicle_tt[i]=.0;

    for( i = 0 ; i < (NumOfLinks+1) ; i++ ) {

        temp = .0;
        for( j=0 ; j< GTTIndex[i] ; j++ ){
            temp += GTT[i][j].TravelTime;
                vehicle_tt[GTT[i][j].VHCID] += GTT[i][j].TravelTime;
        }

            if( GTTIndex[i] != 0){
                temp /= (float) GTTIndex[i];

                total_time += temp;

            }
    }
    total_avg = total_time / NumOfLinks;

        fprintf(outResult, "\n####avg travel time per lane per vehicle = %f\n\n", total_avg);

        for(i=0;i<MAX_VCNT;i++)
            if(vehicle_tt[i]!=.0)
                fprintf(outResult, "travel time per vehicle [%d] = %f\n", i, vehicle_tt[i]);

        fflush(outResult);

        free(vehicle_tt);
}

void got_new_data( struct VHC_TravelTime_s *TT)
{
    float x,y,z,b,g;
    float dist;
    float CurrentTime = qpg_CFG_simulationTime();
    float time_diff;
    float speed;

    static float accumulated_speed=0;
    static int accumulated_cnt=0;
    static int tag=0;

    if( TT->GTT->VHCID % 100 != 0 )
        return;

    // get my position
    qpg_POS_vehicle(global_current_vehicle, global_current_link, &x, &y, &z, &b, &g ); //infospped

    // get distance
    dist = sqrt(pow(x-TT->GTT->x,2) + pow(y-TT->GTT->y,2));  // Calculate Eucleadian distance

    // get time difference
    time_diff = CurrentTime - TT->GTT->ExitTime;

    // calculate info speed
```

```
    if( time_diff > 0) {
        speed = dist / time_diff;

        accumulated_speed += speed;
        accumulated_cnt++;
    }
    else
        speed = -1;

    // print out info speed
    if(speed > 0 )
    {
        fprintf( outResult, "\n%f sender %d receiver %d packet %d dist %f time %f speed %f",
            CurrentTime, TT->GTT->VHCID, qpg_VHC_uniqueID(global_current_vehicle),
            TT->GTT->VHCID*1000000 + TT->GTT->LinkIndex*1000 + ((int)TT->GTT->TravelTime), dist,
time_diff, speed );
        fflush(outResult);
    }

}
```

Appendix B

QualNet code

```
#ifdef _WIN32
#include <winsock2.h>
#else /* unix/linux */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#ifdef sgi
#include <unistd.h>
#endif
#endif


#include <stdio.h>
#include <hash_map>
#include "api.h"
#include "partition.h"
#include "external_util.h"
#include "highway.h"
#include "scheduler.h"
#include "highway_app.h"

#ifdef HIGHWAY_SH
MEMLIB

// shared memory
typedef struct {
    HANDLE shmHandle;
    HANDLE shmMutex;
    LPCTSTR shmBuf;
    int shmSize;
}SHMComm;
SHMComm shmemPQ, shmemQP;

#define SHMBUF_SIZE 2000*1000*16*4

#endif

using namespace stdext;

typedef hash_map<int, Vehicle*> VHASH;


Vehicle *inactive_vehicles = NULL;
Vehicle *active_vehicles = NULL;
Vehicle *neutral_vehicles = NULL;
```

```
int vehicle_pool_cnt=0;
Vehicle vehicles[MAX_VCNT];

VHASH vhash1, vhash2;
VHASH &vhash_neutral = vhash1;
VHASH &vhash_active = vhash2;

Q2PPacket *q2p_pkt;

// global global
FILE *logff;
#ifdef HIGHWAY_STAT
FILE *statff;
#endif

clocktype sync_interval;
float paramics_time_float;
float qualnet_time_float_sec;
clocktype skip_time;

//--------------------------------------------------------------------------
// External Interface API Functions
//--------------------------------------------------------------------------

void HighwayInitializeNodes(
    EXTERNAL_Interface *iface,
    NodeInput *nodeInput)
{
    int           i, j;
    int           channelIndex;
    Node*         nextNode  = NULL;
      struct sockaddr_in addr;
      int                addr_len = sizeof(sockaddr_in);
      HighwayData *data;
    EXTERNAL_SocketErrorType err;

    // Allocate memory for interface-specific data.  The allocated memory
    // is verified by MEM_malloc.  Set the iface->data variable to the
    // newly allocated data for future use.
    data = (HighwayData*) MEM_malloc(sizeof(HighwayData));
    iface->data = (void*) data;

      q2p_pkt = (Q2PPacket *)malloc(MAX_Q2P_PKT_SIZE);

      logff = fopen("log.txt", "w");

#ifdef HIGHWAY_STAT
      statff = fopen("stat.txt", "w" );
#endif

#ifdef STANDALONE
      return;
#endif;


#ifdef HIGHWAY_SHMEMLIB
```

```
    {
    SHMComm SHMCommConnect(char *shmName, int bufSize);
    int SHMCommRead(SHMComm shmComm, char *buf);
    int nRead;


  fprintf(stderr, "Waiting for shared memory ready \n");

     shmemQP = SHMCommConnect("Q2P", SHMBUF_SIZE);
     shmemPQ = SHMCommConnect("P2Q", SHMBUF_SIZE);

  printf("Success with shared memory\n");

    }
#endif

#ifdef HIGHWAY_SOCKET


  // Initialize a listening socket and a data socket
  EXTERNAL_SocketInit(&data->listenSocket);
  EXTERNAL_SocketInit(&data->s);

  // Listen for a socket connection on port 5132.  The newly opened socket
  // connection will be returned in the data->s socket structure.
  printf("Listening for socket connection on port %d...\n", HIGHWAY_PORT);

    err = EXTERNAL_SocketListen(&data->listenSocket, HIGHWAY_PORT, &data->s);
  if (err != EXTERNAL_NoSocketError)
  {
     ERROR_ReportError("Error listening for socket connection");
  }

    if( getsockname(data->s.socketFd, (sockaddr *)&addr, &addr_len) )
    {
        printf("\nerror code=%d", WSAGetLastError() );
        ERROR_ReportError("Error getting address");
    }

    printf("Connection Accepted from %s \n", inet_ntoa(addr.sin_addr));

#endif

#ifdef HIGHWAY_COMMAND


  // Initialize a listening socket and a data socket
  EXTERNAL_SocketInit(&data->listenSocket);
  EXTERNAL_SocketInit(&data->s);

  // Listen for a socket connection on port 5132.  The newly opened socket
  // connection will be returned in the data->s socket structure.
  printf("Listening for socket connection on port %d...\n", HIGHWAY_PORT);

    err = EXTERNAL_SocketListen(&data->listenSocket, HIGHWAY_PORT, &data->s);
  if (err != EXTERNAL_NoSocketError)
```

```
    {
       ERROR_ReportError("Error listening for socket connection");
    }

      if( getsockname(data->s.socketFd, (sockaddr *)&addr, &addr_len) )
      {
           printf("\nerror code=%d", WSAGetLastError() );
           ERROR_ReportError("Error getting address");
      }

      printf("Connection Accepted from %s \n", inet_ntoa(addr.sin_addr));

#endif

      PartitionData* partitionData = iface->partition;


    for (i = 0; i < partitionData->numNodes; i++) {
           HighwaySetInitPosition( iface, i, i*10.0 + 100.0, 1000.0, 0.0 );

      }
      HighwaySetInitPosition( iface, 0, -1000.0, -1000.0, 0.0 );

}


void HighwayReceive(EXTERNAL_Interface *iface)
{
    EXTERNAL_SocketErrorType err;
    HighwayData *data;
    char error[MAX_STRING_LENGTH];
    unsigned int size;
      char simtimebuf[256], realtimebuf[256], paratimebuf[256], bufnext[256];

      clocktype qualnet_time;
      clocktype next_event_time;
      static clocktype paramics_time = 0;
      static clocktype prev_qualnet_time = -1;
      static clocktype start_realtime = EXTERNAL_QueryRealTime();
      static float sent_qualnet_time_float = -1;

      float qualnet_time_float_old, qualnet_time_float_sec;

      qualnet_time = EXTERNAL_QuerySimulationTime(iface);
      qualnet_time_float_sec = qualnet_time / 1000000000.0;
      next_event_time = GetNextInternalEventTime(iface->partition);

      TIME_PrintClockInSecond( qualnet_time, simtimebuf );
      TIME_PrintClockInSecond( EXTERNAL_QueryRealTime()-start_realtime, realtimebuf );
      TIME_PrintClockInSecond( next_event_time, bufnext );

#ifdef FASTSIM
      //SKIP TIME

      if( qualnet_time_float_sec*SECOND < skip_time )
      {
```

```
            fprintf(logff, "\nRecv(): Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec,
realtimebuf );
            printf("\nRecv(): Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec, realtimebuf );

            return;
        }

#endif

#ifndef HIGHWAY_COMMAND

        if( next_event_time < paramics_time_float * SECOND
            || sent_qualnet_time_float >= paramics_time_float )
        {
            return;
        }


#ifdef STANDALONE
        printf("\nSent Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec, realtimebuf );
        fflush(stdout);
        fprintf(logff, "\nSent Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec, realtimebuf );
        fflush(logff);
        return;
#endif

        // use packet

        HighwayPopulatePkt( q2p_pkt, paramics_time_float, iface );

#endif

#ifdef HIGHWAY_SHMEMLIB
{
        int nrecv;
        int SHMCommWrite(SHMComm shmComm, char *buf, int _size);
        int SHMCommRead(SHMComm shmComm, char *buf);

        //if( SHMCommWrite( shmemQP, (char*)&qualnet_time_float_sec, sizeof(qualnet_time_float_sec) ) < 0 )
        //if( SHMCommWrite( shmemQP, (char*)&pkt, sizeof(pkt) ) < 0 )
        if( SHMCommWrite( shmemQP, (char*)q2p_pkt, q2p_pkt->size ) < 0 )
        {
            ERROR_ReportError("\nWrite error");
        }

        sent_qualnet_time_float = q2p_pkt->time;

        printf("\nSent Qualnet Time = %f ---------- REAL TIME = %s", q2p_pkt->time, realtimebuf );
        fflush(stdout);
        fprintf(logff, "\nSent Qualnet Time = %f ---------- REAL TIME = %s", q2p_pkt->time, realtimebuf );
        fflush(logff);

        // Extract the interface-specific data
       data = (HighwayData*) iface->data;

        if( qualnet_time_float_sec * SECOND > iface->partition->maxSimClock - 5 * SECOND ) {
```

```
        HighwayFinalize(iface);
        exit(1);
    }


    do
    {
        nrecv = SHMCommRead(shmemPQ, (char *)&p2q_pkt);
        if ( nrecv < 0 )
            ERROR_ReportError("\nSHMCommRead() Error" );

        else if (nrecv == 0 )
        {
            continue;
        }
        else if (nrecv > 0 )
        {
            paramics_time_float = p2q_pkt.time;

            if( p2q_pkt.vcnt > 0 && paramics_time_float >= qualnet_time_float_sec )
            {
                HighwayUpdateNodeInfo( &p2q_pkt, paramics_time_float*1000000000.0, iface );
            }

            fprintf(logff, "\nQualnet Time=%.10f receive Paramics time=%.10f", qualnet_time_float_sec,
paramics_time_float );
    fflush(logff);
        }

    } while( paramics_time_float <= sent_qualnet_time_float );

}
#endif

#ifdef HIGHWAY_SOCKET

    HighwayForward(iface, (void *) q2p_pkt, q2p_pkt->size );

    printf("\nSent Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec, realtimebuf );
    fflush(stdout);
    fprintf(logff, "\nSent Qualnet Time = %f ---------- REAL TIME = %s", qualnet_time_float_sec, realtimebuf );
    fflush(logff);

    // Extract the interface-specific data
  data = (HighwayData*) iface->data;
    unsigned int size2;

    if( qualnet_time_float_sec * SECOND > iface->partition->maxSimClock - 5 * SECOND ) {
        HighwayFinalize(iface);
        exit(1);
    }

    do
    {
        // read header only
        err = EXTERNAL_SocketRecv(
```

```
        &data->s,
        (char *) &p2q_pkt,
        sizeof(int)+sizeof(float), //256,
        &size,
        FALSE);

    if (err != EXTERNAL_NoSocketError)
    {
        ERROR_ReportError("Error receiving data from socket");
    }

    if( size > 0 ) {
        paramics_time_float = p2q_pkt.time;
    }

    if( size > 0 && p2q_pkt.vcnt > 0 )
    {

        unsigned int to_recv = p2q_pkt.vcnt*sizeof(P2QVehicle);
        unsigned int received = 0;
        char *p = (char *)p2q_pkt.vhcl;

        while( received < to_recv )
        {

                // read data
            err = EXTERNAL_SocketRecv(
                &data->s,
                p + received, // (char *)p2q_pkt.vhcl,
                to_recv-received, //p2q_pkt.vcnt*sizeof(P2QVehicle), //256,
                &size2,
                FALSE);

            if (err != EXTERNAL_NoSocketError)
            {
                ERROR_ReportError("Error receiving data from socket");
            }

            if( size2 < 0 )
                ERROR_ReportError("Error: negative size?" );

            if( size2 > 0 )
                received += size2;

        }

        if( received > 0 )
        {
            assert(received == p2q_pkt.vcnt*sizeof(P2QVehicle));

            if( p2q_pkt.vcnt > 0 && paramics_time_float >= qualnet_time_float_sec )
            {
                HighwayUpdateNodeInfo( &p2q_pkt, paramics_time_float*1000000000.0, iface );
            }
```

```
//                      printf( "\nQualnet Time=%.10f receive Paramis time=%.10f", qualnet_time_float_sec,
paramics_time_float);
                }
                else
                    ERROR_ReportError("Error: header arrived but content didn't come yet?");
            }
    } while( paramics_time_float < qualnet_time_float_sec );

#endif

#ifdef HIGHWAY_COMMAND

   char in[256];
   char payload[256];
     char c;
     int x, y, d;
     Node *node;
   NodeAddress srcNodeId;
   NodeAddress destNodeId;
   NodeAddress srcAddr;
   NodeAddress destAddr;

     // Extract the interface-specific data
   data = (HighwayData*) iface->data;

     IdToNodePtrMap *nodeHash = iface->partition->firstNode->partitionData->nodeIdHash;

     do
     {
          // check packet
          err = EXTERNAL_SocketRecv(
               &data->s,
               in,
               256,
               &size,
               FALSE);

          if (err != EXTERNAL_NoSocketError)
          {
               ERROR_ReportError("Error receiving data from socket");
          }

          if( size > 0 )
          {
          memset(payload, 0, 256);

          sscanf(in, "%c %d %d %d %d", &c, &srcNodeId, &x, &y, &d );

          node = MAPPING_GetNodePtrFromHash(nodeHash, srcNodeId);

               switch(c)
               {
                    case 'g':
                         printf("\nNode %d is at (%f, %f, %f)", srcNodeId,
                              node->mobilityData->current->position.common.c1,
                              node->mobilityData->current->position.common.c2,
```

```
                    node->mobilityData->current->position.common.c3);
                break;
            case 'p':
                printf( "\n got command p node %d (%d %d) at %s", srcNodeId, x, y, simtimebuf );
                HighwayMovePosition( node, x, y, qualnet_time );
                break;
            case 'm':
            {
                printf( "\n got command p: move node %d to (%d %d) during %d seconds at %s",
                    srcNodeId, x, y, d, simtimebuf );
                printf( "\nNot Implemented Yet" );
            }
            case 's':
                printf( "\n got command s at %s", simtimebuf );
                HighwayStartBeacon( node );
                break;
            case 'e':
                printf( "\n got command e at %s", simtimebuf );
                HighwayEndBeacon( node );
                break;
            default:
                break;
        }
        // Get node addresses
        srcAddr = MAPPING_GetDefaultInterfaceAddressFromNodeId(
            iface->partition->firstNode,
            srcNodeId);

        // Verify valid pointers
        if (srcAddr == INVALID_MAPPING )
        {
            ERROR_ReportWarning("Invalid address for interfacetutorial");
            continue;
        }

        }
    } while( size > 0 );

#endif

}

void HighwayForward(
    EXTERNAL_Interface *iface,
    void *forwardData,
    int forwardSize)
{
    EXTERNAL_SocketErrorType err;
    HighwayData *data;

    // Extract interface-specific data
    data = (HighwayData*) iface->data;

#ifdef HIGHWAY_SOCKET

    // Send forwarded information on the data socket
```

```
    err = EXTERNAL_SocketSend(
        &data->s,
        (char*) forwardData,
        forwardSize);
    if (err != EXTERNAL_NoSocketError)
    {
        ERROR_ReportError("Error sending data on socket");
    }
#endif
}

void HighwayFinalize(EXTERNAL_Interface *iface)
{
    HighwayData *data;
    EXTERNAL_SocketErrorType err;

      fclose(logff);
#ifdef HIGHWAY_STAT
      fclose(statff);
#endif

      if( q2p_pkt )
           free(q2p_pkt);

    // Extract interface-specific data
    data = (HighwayData*) iface->data;

#ifdef HIGHWAY_SHMEM
      // unmap shaered memory
      UnmapViewOfFile(shmBuf);

      CloseHandle(shmMapFile);
      // end of shared memory
#endif

#ifdef HIGHWAY_SOCKET

    // Close the data socket
    err = EXTERNAL_SocketClose(&data->s);
    if (err != EXTERNAL_NoSocketError)
    {
        ERROR_ReportError("Error closing socket");
    }

    // Close the listening socket
    err = EXTERNAL_SocketClose(&data->listenSocket);
    if (err != EXTERNAL_NoSocketError)
    {
        ERROR_ReportError("Error closing socket");
    }
#endif
}

void HighwaySimulationHorizon(EXTERNAL_Interface *iface)
{
#ifdef FASTSIM
```

```
    clocktype tempHorizon;

    if( iface->partition->theCurrentTime < skip_time-10*SECOND )
        iface->horizon = skip_time;
    else
    {
        static clocktype starttime = EXTERNAL_QueryRealTime();

        clocktype realhorizon = EXTERNAL_QueryRealTime()-starttime; //iface->lookahead;

        if( iface->horizon < paramics_time_float * SECOND )
        {
                iface->horizon = paramics_time_float * SECOND; // test. follow paramics

            /* debug
            char bufhorizon[256], buftime[256], bufnext[256];
            TIME_PrintClockInSecond( iface->horizon, bufhorizon );
            TIME_PrintClockInSecond( iface->partition->theCurrentTime, buftime );
            TIME_PrintClockInSecond( GetNextInternalEventTime(iface->partition), bufnext );

            if( paramics_time_float > 0 )
                fprintf(logff, "\nHorizon=%s paramics_time=%f currenttime=%s, nextevent=%s",
                        bufhorizon, paramics_time_float,buftime, bufnext );
            */
        }

        // just in casae it's slower than realtime, follow real time
        if( iface->horizon < realhorizon )
            iface->horizon = realhorizon;

//      else
//          iface->horizon = iface->partition->theCurrentTime;

        return;
    }

#endif

}

// This works only at initialization
// In the middle of simulation, use HighwayMovePosition()
void HighwaySetInitPosition( EXTERNAL_Interface *iface, int node,
                    double c1, double c2, double c3 )
{
    iface->partition->nodePositions[node].mobilityData->current->position.common.c1 = c1;
    iface->partition->nodePositions[node].mobilityData->current->position.common.c2 = c2;
    iface->partition->nodePositions[node].mobilityData->current->position.common.c3 = c3;
}

void HighwayMovePosition( Node *node, double x, double y, clocktype time )
{
    Coordinates position;
    Orientation orientation;
    position.common.c1 = x;
```

```
    position.common.c2 = y;
    position.common.c3 = 0;
    orientation.azimuth = orientation.elevation = 0;

    MOBILITY_AddANewDestination(
    node->mobilityData,
    time,
    position,
    orientation);

    MobilityData *mobilityData = node->mobilityData;
    MobilityRemainder *remainder = &(node->mobilityData->remainder);

    memcpy( mobilityData->next, &mobilityData->destArray[mobilityData->numDests-1], sizeof(MobilityElement)
);
    mobilityData->sequenceNum++;
    mobilityData->next->sequenceNum = mobilityData->sequenceNum;

    remainder->nextMoveTime = time;
    remainder->nextPosition = mobilityData->next->position;
    remainder->nextOrientation = mobilityData->next->orientation;
    remainder->speed = mobilityData->next->speed;
    remainder->numMovesToNextDest = 0;
    remainder->destCounter = mobilityData->numDests-1;

    MOBILITY_InsertEvent(&(node->partitionData->mobilityHeap), node);
    MOBILITY_ProcessEvent(node);

}

void HighwayStartBeacon( Node *node )
{
    AppDataHighway *appData;
    Message *timerMsg;
    AppTimer *timer;

    appData = AppHighwayGet(node, 100);
    appData->running = true;
    timerMsg = MESSAGE_Alloc(node,
                             APP_LAYER,
                             APP_HIGHWAY,
                             MSG_APP_TimerExpired);

    MESSAGE_InfoAlloc(node, timerMsg, sizeof(AppTimer));

    timer = (AppTimer *)MESSAGE_ReturnInfo(timerMsg);

    timer->sourcePort = appData->srcPort;
    timer->type = APP_TIMER_SEND_PKT;

    MESSAGE_Send(node, timerMsg, 0); // start now
}

void HighwayEndBeacon( Node *node )
{
    AppDataHighway *appData;
```

```
    appData = AppHighwayGet(node, 100);
    appData->running = false;
}

void HighwayUpdateNodeInfo( P2QPacket *p, clocktype t, EXTERNAL_Interface *iface )
{
    vehicle_start_update();

    for( int i=0; i < p->vcnt; i++ )
    {
        Vehicle *v = vehicle_get( p->vhcl[i].vid ); // should have non null node/appData

        assert(v->node);

        HighwayMovePosition(v->node, p->vhcl[i].x, p->vhcl[i].y, t );

        if( p->vhcl[i].pkt_size < 0 )
        {
            fprintf(logff, "\nERROR: P->Q sent negative packet size (%d) for vid %d", p->vhcl[i].pkt_size, p-
>vhcl[i].vid );
            p->vhcl[i].pkt_size = 100;
        }
        v->appData->pktSize = p->vhcl[i].pkt_size;
    }

    vehicle_end_update();

    if(0)
    {
        fprintf(logff, "\n[P->Q: time=%f, vcnt=%d]", p->time, p->vcnt );
        for( int i=0; i < p->vcnt; i++ )
        {
            fprintf(logff, "\n\t[vid=%d, pkt_size=%d, position=(%f,%f)]", p->vhcl[i].vid, p->vhcl[i].pkt_size, p-
>vhcl[i].x, p->vhcl[i].y );
        }
    }

}

void HighwayPopulatePkt( Q2PPacket *p, float t, EXTERNAL_Interface *iface )
{
    p->time = t;
    p->vcnt = 0;
    p->size = 2*sizeof(int) + sizeof(float);

    // END OF SIMULATION
    if( t * SECOND > iface->partition->maxSimClock - 5 * SECOND ) {
        p->time = -1;
        return;
    }


    Q2PVehicle *vp = p->vhcl;

    for( Node *np=iface->partition->firstNode; np; np = np->nextNodeData )
```

```
    {
        AppDataHighway *appData = AppHighwayGet(np, HIGHWAY_APP_PORT );

        if( appData && appData->running && appData->q2p_vehicle.rcnt > 0 )
        {
            vp->rvid = appData->p_vid; //np->nodeId;
            vp->rcnt = appData->q2p_vehicle.rcnt;
            memcpy( vp->rcv, appData->q2p_vehicle.rcv, vp->rcnt * sizeof(Q2PRecv) );
            p->vcnt++;
            p->size += 2*sizeof(int) + vp->rcnt * sizeof(Q2PRecv);
            vp = (Q2PVehicle *)&vp->rcv[vp->rcnt];
            appData->q2p_vehicle.rcnt = 0;
        }
    }

    if(0)
    {
        fprintf(logff, "\n[Q->P: time=%f, vcnt=%d, size=%d]", p->time, p->vcnt, p->size );
        vp = p->vhcl;
        for( int i=0; i < p->vcnt; i++ )
        {
            fprintf(logff, "\n\t[rvid=%d, rcnt=%d]", vp->rvid, vp->rcnt );
            for( int j=0; j < vp->rcnt; j++ )
            {
                fprintf(logff, "\n\t\t[svid=%d, rtime=%f]", vp->rcv[j].svid, vp->rcv[j].rtime );
            }
            vp = (Q2PVehicle *)&vp->rcv[vp->rcnt];
        }
    }

}

////////////////////////////////////////
// Vehicles functions
////////////////////////////////////////

// add a node to the pool of vehicles
// inactivate
void vehicle_add( Node *node, AppDataHighway *data )
{
    Vehicle *v;

    if( vehicle_pool_cnt >= MAX_VCNT - 1)
    {
        printf( "no more memory for vehicle" );
        exit(1);
    }

    if( node->nodeId == 1 )
        return;

    v = &vehicles[vehicle_pool_cnt++];

    v->node = node;
    v->appData = data;
```

```
    // add to inactive_vehicles
    v->next = inactive_vehicles;
    v->prev = NULL;
    if( inactive_vehicles )
        inactive_vehicles->prev = v;
    inactive_vehicles = v;
}

// move one active vehicle
// to inactive vehicle list
void vehicle_inactivate(Vehicle *v)
{
    // remove from active list
    if( v->prev )
        v->prev->next = v->next;
    if( v->next )
        v->next->prev = v->prev;
    if( v == active_vehicles )
        active_vehicles = v->next;

    // add to inactive_vehicles
    v->next = inactive_vehicles;
    v->prev = NULL;
    if( inactive_vehicles )
        inactive_vehicles->prev = v;
    inactive_vehicles = v;
}

// move one inactive vehicle
// to active vehicle list
Vehicle *vehicle_activate()
{
    Vehicle *v;

    if( !inactive_vehicles )
    {
        ERROR_ReportError( "no more inactive vehicle to activate" );
    }

    // remove from inactive vehicle
    v = inactive_vehicles;
    inactive_vehicles = inactive_vehicles->next;
    inactive_vehicles->prev = NULL;

    // add to active_vehicles
    v->next = active_vehicles;
    v->prev = NULL;
    if( active_vehicles )
        active_vehicles->prev = v;

    return v;
}

Vehicle *vehicle_get( int p_vid )
{
    Vehicle *v;
```

```
    hash_map <int, Vehicle *> :: iterator itr;

    // get existing one
    itr = vhash_neutral.find( p_vid );

    // new vehicle arrived
    if( itr == vhash_neutral.end() )
        v = vehicle_activate();
    else
    {
        v = itr->second;

        // remove from neutral_vehicles
        if( v->prev )
            v->prev->next = v->next;
        if( v->next )
            v->next->prev = v->prev;
        if( v == neutral_vehicles )
            neutral_vehicles = v->next;

        // remove from vhash_neutral
        vhash_neutral.erase(itr);

        // add to active_vehicles
        v->next = active_vehicles;
        v->prev = NULL;
        if( active_vehicles )
            active_vehicles->prev = v;
        active_vehicles = v;
    }

    // add to vhash_active
    vhash_active[p_vid] = v;

    // this is actually new car released
    if( v->appData->running == false )
    {
        v->appData->running = true;
        v->appData->p_vid = p_vid;

        AppHighwayScheduleNextPkt(v->node, v->appData);
    }
    else
        v->appData->p_vid = p_vid;

    return v;
}

// move all vehicles from active_list to neutral_list
void vehicle_start_update()
{
    neutral_vehicles = active_vehicles;
    active_vehicles = NULL;

    VHASH &temp = vhash_neutral;
    vhash_neutral = vhash_active;
```

```
    vhash_active = temp;

    vhash_active.clear();
}

void vehicle_end_update()
{
    // for each vehicle in neutral_list
    for(Vehicle *v=neutral_vehicles; v; )
    {
        Vehicle *t = v->next;

        // add to inactive_vehicles
        v->next = inactive_vehicles;
        v->prev = NULL;
        if( inactive_vehicles )
            inactive_vehicles->prev = v;
        inactive_vehicles = v;
        v->appData->running = false;

        v = t;
    }
    neutral_vehicles=NULL;
}


#ifdef HIGHWAY_SHMEMLIB

// -------------------------- Shared Mem Communication API DEFINITIONS -------------------------//
FILE *errout = stdout;

#if 0
// create shared memory region
// returns when it accepts a connection
SHMComm SHMCommCreate(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    shmComm.shmHandle = CreateFileMapping(
                        INVALID_HANDLE_VALUE,    // use paging file
                        NULL,                    // default security
                        PAGE_READWRITE,          // read/write access
                        0,                       // max. object size
                        bufSize,                 // buffer size
                        shmName);

    if (shmComm.shmHandle == NULL)
    {
        fprintf(errout, "Could not create file mapping object (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
                FILE_MAP_ALL_ACCESS, // read/write permission
```

```
                0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout,"Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    memset((char*)shmComm.shmBuf, 0, bufSize);

    *((int *)shmComm.shmBuf) = 0;
    *((int *)(shmComm.shmBuf + sizeof(int))) = 0;

    sprintf(name_buf, "%sMutex", shmName);
    // create mutex
    shmComm.shmMutex = CreateMutex(
              NULL,     // default security attributes
                FALSE,  // initially not owned
                name_buf);

    if (shmComm.shmMutex == NULL)
    {
        fprintf(errout,"Could not create mutex lock (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    return shmComm;

}


SHMComm SHMCommConnect(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    // !!! infinite loop to open shared memory
    while( NULL == (shmComm.shmHandle = OpenFileMapping(
                FILE_MAP_ALL_ACCESS,   // read/write access
                FALSE,                 // do not inherit the name
                shmName))) Sleep(1000);

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
              FILE_MAP_ALL_ACCESS, // read/write permission
              0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout, "Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    // !!! another infinite loop to open mutex

    sprintf(name_buf, "%sMutex", shmName);
```

```
    while ( NULL == (shmComm.shmMutex = OpenMutex(
                        MUTEX_ALL_ACCESS,      // request full access
                        FALSE,                 // handle not inheritable
                        name_buf))) Sleep(1000);

    shmComm.shmSize = bufSize;

    return shmComm;
}
#endif

// create shared memory region
// returns when it accepts a connection
SHMComm SHMCommCreate(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    shmComm.shmHandle = CreateFileMapping(
                    INVALID_HANDLE_VALUE,    // use paging file
                    NULL,                    // default security
                    PAGE_READWRITE,          // read/write access
                    0,                       // max. object size
                    bufSize,                 // buffer size
                    shmName);

    if (shmComm.shmHandle == NULL)
    {
        fprintf(errout, "Could not create file mapping object (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
                FILE_MAP_ALL_ACCESS, // read/write permission
                0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout,"Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    memset((char*)shmComm.shmBuf, 0, bufSize);

    *((int *)shmComm.shmBuf) = 0;
    *((int *)(shmComm.shmBuf + sizeof(int))) = 0;

    sprintf(name_buf, "%sMutex", shmName);
    // create mutex
    shmComm.shmMutex = CreateMutex(
                NULL,     // default security attributes
                  FALSE,  // initially not owned
                  name_buf);
```

```
    if (shmComm.shmMutex == NULL)
    {
        fprintf(errout,"Could not create mutex lock (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    shmComm.shmSize = bufSize;

    return shmComm;

}


SHMComm SHMCommConnect(char *shmName, int bufSize){
    SHMComm shmComm;
    char name_buf[30];

    // !!! infinite loop to open shared memory
    while( NULL == (shmComm.shmHandle = OpenFileMapping(
                FILE_MAP_ALL_ACCESS,   // read/write access
                FALSE,                 // do not inherit the name
                shmName))) Sleep(1000);

    // get a buf pointer after mapping shm
    shmComm.shmBuf = (LPTSTR) MapViewOfFile(shmComm.shmHandle,   // handle to map object
                FILE_MAP_ALL_ACCESS, // read/write permission
                0, 0, bufSize);
    if (shmComm.shmBuf == NULL)
    {
        fprintf(errout, "Could not map view of file (%d).\n", GetLastError());
        fflush(errout);
        exit(1);
    }

    sprintf(name_buf, "%sMutex", shmName);
    // !!! another infinite loop to open mutex
    while ( NULL == (shmComm.shmMutex = OpenMutex(
                            MUTEX_ALL_ACCESS,      // request full access
                            FALSE,                 // handle not inheritable
                            name_buf))) Sleep(1000);

    shmComm.shmSize = bufSize;

    return shmComm;
}


int SHMCommWrite(SHMComm shmComm, char *buf, int _size) {
    DWORD waitResult;
    int head, rear;  // head, rear of circular queue
    char *cq;           // circular queue
    int cq_size; // circular queue size
    int first_half, second_half;
    int data_size, block_size;
```

```
    if(NULL == buf) {
        fprintf(errout, "ERROR - SHMCommWrite: buf is NULL..\n");
        fflush(errout);
        return -1;
    }

    // if data size is not divided by 4, we append some nulls
    if(0 != _size%4) {
        data_size = _size + (4 - _size%4);
        fprintf(errout, "WARNING - SHMCommWrite: size is not divided by 4. \n");
        fflush(errout);
    }
    else
        data_size = _size;

    block_size = data_size + sizeof(int);

LABEL:
    while(1){
        waitResult = WaitForSingleObject(
                        shmComm.shmMutex,   // handle to mutex
                        5000L);   // five-second time-out interval
        if(waitResult == WAIT_OBJECT_0)
            break; // got mutex lock
    }

    // now mutual exclusion block starts from here

    // first 4 byte points to the head of circular queue
    // second 4 byte points to the rear of circular queue where new data should be appended
    head = *((int *)shmComm.shmBuf);
    rear = *((int *)(shmComm.shmBuf + sizeof(int)));
    cq = (char*) (shmComm.shmBuf + 2*sizeof(int));
    cq_size = shmComm.shmSize - 2*sizeof(int);


    if(rear < head) {
        fprintf(errout, "TEST1\n");
        fprintf(errout, "rear = %d < head = %d\n", rear, head);
        fflush(errout);

        // check whether new rear would exceed head
        if( (rear+block_size)>=head ) {
            fprintf(errout, "ERROR - SHMCommWrite: Shared memory buffer is filled up.\n");
            fflush(errout);

            ReleaseMutex(shmComm.shmMutex);
            exit(1);
        }

        memcpy(cq+rear, &data_size, sizeof(int));

        rear = (rear+block_size)%cq_size;
    }
    else if( (rear+block_size) > cq_size ){
        fprintf(errout, "TEST2\n");
```

```
            fprintf(errout, "head = %d\n", head);
            fprintf(errout, "rear = %d\n", rear);

            fflush(errout);

            // need to wrap around
            // check whether new rear would exceed head
            if( (rear + block_size - cq_size) >= head ) {
                fprintf(errout, "ERROR - SHMCommWrite: Shared memory buffer is filled up.\n");
                fflush(errout);
                ReleaseMutex(shmComm.shmMutex);
                goto LABEL;
            }

            memcpy(cq+rear, &data_size, sizeof(int));

            first_half = cq_size - (rear + sizeof(int));
            second_half = _size - first_half;

            memcpy(cq+rear+sizeof(int), buf, first_half);
            memcpy(cq, buf + first_half, second_half);

            rear = ( rear + block_size ) % cq_size;
        }
        else {
            // nothing to worry about
            memcpy(cq+rear, &data_size, sizeof(int));
            memcpy(cq+rear+sizeof(int), buf, _size);

            rear = ( rear + block_size ) % cq_size;
        }
        fprintf(errout, "----\n");


        *((int *)(shmComm.shmBuf + sizeof(int))) = rear;

        // mutual exclusion block ends here

        if (! ReleaseMutex(shmComm.shmMutex)) {
            fprintf(errout, "\n0) Error for release (%d)\n", GetLastError() );
            fflush(errout);
            return -1;
        }

        return block_size;
    }

    int SHMCommRead(SHMComm shmComm, char *buf) {
        DWORD waitResult;
        int head, rear;  // head, rear of circular queue
        char *cq;           // circular queue
        int cq_size; // circular queue size
        int first_half, second_half;
        int size;

        if(NULL == buf) {
```

```
        fprintf(errout, "ERROR - SHMCommRead: buf is NULL..\n");
        fflush(errout);
        return -1;
    }

        while(1){
            waitResult = WaitForSingleObject(
                                shmComm.shmMutex,   // handle to mutex
                                5000L);   // five-second time-out interval
            if(waitResult== WAIT_OBJECT_0)
                break; // got mutex lock
        }

        // now mutual exclusion block starts from here

        // first 4 byte points to the head of circular queue
        // second 4 byte points to the rear of circular queue where new data should be appended
        head = *((int *)shmComm.shmBuf);
        rear = *((int *)(shmComm.shmBuf + sizeof(int)));
        cq = (char*) (shmComm.shmBuf + 2*sizeof(int));
        cq_size = shmComm.shmSize - 2*sizeof(int);

        if(head == rear) {
            if (! ReleaseMutex(shmComm.shmMutex)) {
                fprintf(errout, "\n2) Error for release (%d)\n", GetLastError() );
                fflush(errout);
                return -1;
            }
            return 0;
        }

    size = *((int *)(cq+head));
    if( (head + (int)sizeof(int) + size) > cq_size){
        // wrap around
        first_half = cq_size - head - sizeof(int);
        second_half = size - first_half;

        memcpy(buf, cq + head + sizeof(int), first_half);
        memcpy(buf+first_half, cq, second_half);

        head = ( head + size + sizeof(int) )% cq_size;
    }
    else {
        // nothing to worry about
        memcpy(buf, cq+head+sizeof(int), size);
        head = ( head + size + sizeof(int) )% cq_size;
    }

    *((int *)shmComm.shmBuf) = head;

    // mutual exclusion block ends here

    if (! ReleaseMutex(shmComm.shmMutex)) {
        fprintf(errout, "\n1) Error for release (%d)\n", GetLastError() );
        fflush(errout);
        return -1;
```

```
    }

    return size;
}


void SHMCommClose(SHMComm shmComm){
    // unmap shaered memory
    UnmapViewOfFile(shmComm.shmBuf);

    CloseHandle(shmComm.shmHandle);
    // end of shared memory
}


#endif
```

Abbreviations

ACK	Acknowledgement
AP	Access Point
API	Application Programming Interface
CDMA	Code Division Multiple Access
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access / Collision Detect
CTS	Clear-To-Send
DARPA	Defense Advanced Research Projects Agency
DCF	Distributed Coordination Function
DGPS	Differential Global Positioning System
DIFS	DCF Inter-Frame Spacing
DOLPHIN	Dedicated Omni–purpose inter–vehicle communication Linkage Protocol for Highway automation
DSRC	Dedicated Short Range Communications
E2E	End–to–End
ECE	Electrical and Computer Engineering
FDMA	Frequency Division Multiple Access
FHSS	Frequency-Hopping Spread Spectrum
FHWA	Federal Highway Administration
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronic Engineering

IP	Internet Protocol
ISM	Industrial/Scientific/Medical
ISO	International Organization for Standardization
ISR	Institutes for Systems Research
ITS	Intelligent Transportation Systems
IVC	Inter–Vehicle Communications
MAC	Medium Access Control
MANET	Mobile ad hoc network
MOP	Measures of Performance
OBU	Onboard Unit
OSI	Open Systems Interconnection
PCF	Point Coordination Function
PDA	Personal Digital Assistants
PRNET	Packet Radio network
RMDP	Received Message-Dependent Protocol
RSU	Roadside Unit
RTE	Real Time Extension
RTS	Request-To-Send
SIG	Special Interest Group
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TMC	Traffic Monitoring Center
UDP	User Datagram Protocol

US DOTU.S.	Department of Transportation
VANET	Vehicular ad hoc network
VII	Vehicle Infrastructure Integration
Wi-Fi	Wireless Fidelity
WLAN	Wireless Local Area Networks

References

1. Abramson, N. (1970). The ALOHA system–Another alternative for computer communications. Proceedings of the AFIPS Conference, Fall Joint Computer Conference, 37, pp. 281-285.
2. Abramson, N. (1985). Development of the ALOHANET. IEEE Transactions on Information Theory, 31(2), pp. 119-123.
3. AIMSUM Homepage. (2007). Transport Simulation Systems, Spain. www.aimsun.com, accessed October 29, 2007.
4. Al-Deek, H., and Chandra, C. (2004). New algorithms for filtering and imputation of real-time and archived dual-loop detector data in I-4 data warehouse. In Transportation Research Record: Journal of the Transportation Research Board, No. 1867, TRB, National Research Council, Washington, D.C., pp. 116-126.
5. Avila, A., Korkmaz, G., Liu,Y., Teh, H., Ekici, E., Ozguner, F., Ozguner, U., Redmill, K., Takeshita, O., Tokuda, K., Hamaguchi, M., Nakabayashi, S., and Tsutsui, H. (2005). A complete simulator architecture for inter-vehicle communication based intersection warning system. Proceedings of the 8th IEEE Conference on ITS, Vienna, Austria, pp. 461–466.
6. Blum, J., Eskandarian, A., and Hoffman, L. (2004). Challenges of intervehicle *ad hoc* networks. IEEE Transactions on ITS, 5(4), pp. 347–351.
7. Bogenberger, R., and Kosch, T. (2002). Ad-hoc peer-to-peer communication - Webs on the street. Proceedings of the 9th World Congress on ITS, Chicago, Illinois.

8. Broadcom (2006). 802.11n: Next-generation wireless LAN technology. White paper, Broadcom Corporation, Irvine, CA. www.broadcom.com/docs/WLAN/802_11n-WP100-R.pdf, accessed October 29, 2007.

9. Chawathe, S. S. (2006). Inter-vehicle data dissemination in sparse equipped traffic. Proceedings of the 9th IEEE Conference on ITS, Toronto, Canada, pp. 273–280.

10. Chen, C., Kwon, J., Rice, J., Skabardonis, A., and Varaiya, P. (2003). Detecting errors and imputing missing data for single-loop surveillance systems. In Transportation Research Record: Journal of the Transportation Research Board, No. 1855, TRB, National Research Council, Washington, D.C., pp. 160-167.

11. Chen, M., and Chien, S. I. J. (2000). Determinating the number of probe vehicles for freeway travel time estimation using microscopic simulation. In Transportation Research Record: Journal of the Transportation Research Board, No. 1719, TRB, National Research Council, Washington D.C., pp. 61–68.

12. Chen, Z., Kung, H., and Vlah, D. (2001). Ad hoc relay wireless networks over moving vehicles on highways. Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking & Computing, Long beach, California, pp. 247-250.

13. Chisalita, L., and Shahmehri, N. (2002). A peer-to-peer approach to vehicular communication for the support of traffic safety application. Proceedings of the 5th IEEE Conference on ITS, Singapore, pp. 336-341.

14. Corsim Homepage. (2006). Federal Highway Administration, Washington D. C. ops.fhwa.dot.gov/trafficanalysistools/corsim.htm, accessed October 29, 2007.

15. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1, Springer Berlin, Heidelberg, Germany. pp. 269–271.
16. Dogan, A., Korkmaz, G., Liu, Y., Ozguner, F., Ozguner, U., Redmill, K., Takeshita, O., and Tokuda, K. (2004). Evaluation of intersection collision warning system using an inter-vehicle communication simulator. Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 1103-1108.
17. FleetNet (2002). FleetNet - Internet on the road. www.et2.tu-harburg.de/fleetnet/pdf/FleetNet_Flyer.pdf, accessed October 29, 2007.
18. Frodigh, M., Johansson, P., and Larsson, P. (2000). Wireless ad hoc networking: The art of networking without a network. Ericsson Review, 4, pp. 248-263.
19. GloMoSim Homepage. (2001). University of California at Los Angeles, UCLA Parallel Computing Laboratory, California. pcl.cs.ucla.edu/projects/glomosim, accessed October 29, 2007.
20. Goel, S., Imielinski, T., and Ozbay, K. (2004). Ascertaining viability of WiFi based vehicle-to-vehicle network for traffic information dissemination. Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 1086-1091.
21. Gold, D., Turner, S., Gajewski, B., and Spiegelman, C. (2001). Imputing missing values in ITS data archives for intervals under 5 minutes. Proceedings of the 80th TRB Annual Meeting, Washington, D.C.
22. Hasegawa, T., Mizui, K., Fujii, H., and Seki, K. (2004). A concept reference model for inter-vehicle communications (Report 2). Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 810-815.

23. ITS America. (2005). Primer on Vehicle-Infrastructure Integration. VII White Paper Series, www.itsa.org/itsa/files/pdf/VIIPrimer.pdf, accessed October 29, 2007.

24. Jubin, J., and Tornow, J. D. (1987). The DARPA packet radio network protocol. Proceedings of the IEEE, 75(1), pp. 21-32.

25. Kim, H., and Lovell, D. (2006a). Determining Spatio-Temporal Limits of Traffic Information for Imputation in Vehicular Ad Hoc Networks. Proceedings of the 13th World Congress on ITS, London, United Kingdom.

26. Kim, H., and Lovell, D. (2006b). Traffic information imputation using a linear model in vehicular ad hoc networks. Proceedings of the 8th IEEE Conference on ITS, Toronto, Canada, pp. 1406-1411.

27. Kim, H., Lovell, D., and Kim, T. (2007a). Reliable Range of Individual Travel Time Information in Vehicular Ad hoc Networks. Proceedings of the 86th TRB Annual Meeting, Washington D.C., USA.

28. Kim, H., Lovell, D., Kang, Y., and Kim, W. (2007b). Data Quantity for Travel Time Estimation in Vehicular Ad Hoc Networks. Proceedings of the 66th IEEE Conference on Vehicular Technology, Baltimore, USA.

29. Leung, K., K, Y., Dao, T., Clark, C. M., and Huissoon, J. P. (2006). Development of a microscopic traffic simulator for inter-vehicle communication application research. Proceedings of the 9th IEEE Conference on ITS, Toronto, Canada, pp. 1286–1291.

30. Little, D. C. T., and Agarwal, A. (2005). An information propagation scheme for VANETs. Proceedings of the 8th IEEE Conference on ITS, Vienna, Austria, pp. 155–160.
31. Liu, Y., F. Dion, and Biswas, S. (2005). Dedicated Short-range wireless communications for intelligent transportation system applications – state of the art. In Transportation Research Record: Journal of the Transportation Research Board, No. 1910, TRB, National Research Council, Washington, D.C., pp.29–37.
32. Metcalfe, R. (1973). Packet communication. Ph.D. Dissertation, MIT, Cambridge, Massachusetts.
33. Murthy, C., and Manoj, B (2004). Ad hoc wireless networks: architectures and protocols. Prentice Hall, New Jersey, USA.
34. Nadeem, T., Dashtinezhad, S., and Liao, C. (2004). Traffic view: A scalable traffic monitoring system. Proceedings of the IEEE International Conference on Mobile Data Management, Berkeley, California, pp. 1-14.
35. Newell, G. (1993). A simplified theory of kinematic waves in highway traffic: I general theory, II queuing at freeway bottlenecks and III multi-destination flows. Transportation Research B, 27(4), pp. 281-313.
36. NS-2 Homepage. (2007).University of Southern California, Information Sciences Institute, California. www.isi.edu/nsnam/ns, accessed October 29, 2007.
37. OPNET Homepage. (2007). OPNET Technologies, Inc., Maryland. www.opnet.com, accessed October 9, 2007.

38. Ott, J., and Kutscher, D. (2004). Drive-thru Internet: IEEE 802.11b for "automobile" users. Proceedings of the 23rd Annual joint Conference of the IEEE Computer and Communications Societies, Hong Kong, China.
39. Paramics Homepage. (2007). Quadstone Paramics, United Kingdom. paramics-online.com, accessed October 29, 2007.
40. QualNet Homepage. (2007). Scalable Network Technologies, California. www.scalable-networks.com, accessed October 29, 2007.
41. Ramanathan, R., and Redi, J. (2002). A brief overview of ad hoc networks: challenges and directions. IEEE Communications Magazine, 50th Anniversary Commemorative Issue, pp. 20-22.
42. Saito, M., Tsukamoto, J., Umedu, T., and Higashino, T. (2007). Design and Evaluation of Intervehicle Dissemination Protocol for Propagation of Preceding Traffic Information. IEEE Transactions on ITS, 8(3), pp. 379-390.
43. Sawant, H., Tan, J., Yang, Q., and Wang, Q. (2004). Using Bluetooth and sensor networks for Intelligent Transportation Systems. Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 767-772.
44. Smith, B., Scherer, W., and Conklin, J., (2003). Exploring imputation techniques for missing data in Transportation Management Systems. In Transportation Research Record: Journal of the Transportation Research Board, No. 1836, TRB, National Research Council, Washington, D.C., pp. 132-142.
45. Srinivasan, K. K., and Jovanis, P. P. (1996). Determination of the number of probe vehicles required for reliable travel time measurement in urban network.

Transportation Research Record 1537, TRB, National Research Council, Washington D.C., pp. 15–22.

46. Tan, G., Miu, A., Guttag, J., and Balakrishnan, H. (2001). Forming scatternets from Bluetooth personal area networks. MIT Technical Report, MIT-LCS-TR-826, Cambridge, Massachusetts.
47. Tokuda, K., Akiyama, M., and Fujii H. (2000). DOLPHIN for inter-vehicle communications system. Proceedings of the IEEE Intelligent Vehicles Symposium 2000, Dearborn, Michigan, pp. 504-509.
48. Toppen, A., and Wunderlich, K. (2004). Travel time data collection for measurement of advanced traveler information systems accuracy. Proceedings of the14th ITS America Annual Meeting, San Antonio, Texas.
49. Ueki, J., Mori, J., Nakamura, Y., Horii, Y., and Okada, H. (2004). Development of vehicular-collision avoidance support system by inter-vehicle communications –VCASS –. Proceedings of the 59th IEEE Conference on Vehicular Technology Spring, 5, Milan, Italy, pp. 2940–2945.
50. US DOT (2007). ITS joint program office home. www.its.dot.gov, accessed October 29, 2007.
51. VISSIM Homepage. (2007). Planung Transport Verkehr AG, Germany. www.english.ptv.de/cgi-bin/traffic/traf_vissim.pl, accessed October 29, 2007.
52. Wang, L., Wang, C., Shen, X., and Fan, Y. (2005). Probe vehicle sampling for real-time traffic data collection. Proceedings of the 8th IEEE Conference on ITS, Vienna, Austria, pp. 886-888.

53. Wang, S. (2004). On the intermittence of routing paths in vehicle-formed mobile ad hoc networks on highways. Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 803-809.
54. Werner, J. (2004). More details emerge about the VII effort. Newsletter of the ITS cooperative deployment network, www.ntoctalks.com/icdn/vii_details_itsa04.html, accessed October 29, 2007.
55. Werner, J. (2005). Details of the VII initiative's 'Work in Progress' provided at public meeting. www.ntoctalks.com/icdn/vii_pubmtg_v1.php, accessed October 29, 2007.
56. Wischhof, L., Ebner, A., and Rohling, H. (2005). Information dissemination in self-organizing intervehicle networks. IEEE Transactions on ITS, 6(1), pp. 90-101.
57. Wu, H. (2005). Analysis and design of vehiclualr networks. Ph. D. dissertation, Georgia Institute of Technology, Atlanta, US.
58. Wu, H., Lee, J., Hunter, M., Fujimoto, R., Guensler, R., and Ko, J. (2005). Efficiency of simulated vehicle-to-vehicle message propagation in Atlanta, Georgia, I-75 corridor. In Transportation Research Record: Journal of the Transportation Research Board, No. 1910, TRB, National Research Council, Washington, D.C., pp.82–89.
59. Xu, H., and Barth, M. (2006).Travel time estimation techniques for traffic information systems based on intervehicle communications. In Transportation Research Record: Journal of the Transportation Research Board, No. 1944, TRB, National Research Council, Washington, D.C., pp.72–81.
60. Xu, Q., Mak, T., Ko, J., and Sengupta, R. (2004). Layer-2 protocol design for vehicle safety communications in dedicated short range communications

spectrum. Proceedings of the 7th IEEE Conference on ITS, Washington, D.C., pp. 1092-1097.

61. Yang, X. (2003). Assessment a self-organizing distributed traffic information system: modeling and simulation. Ph. D. dissertation, University of California, Irvine, US.
62. Yin, J., Elbatt, T., Yeung, G., Ryu, B., Habermas, S., Krishnan, H., and Talty, T. (2004). Performance evaluation of safety applications over DSRC vehicular ad hoc networks. Proceedings of the 1st ACM Workshop on Vehicular Ad hoc Networks, Philadelphia, Pennsylvania, pp. 1–9.
63. Ziliaskopoulos, A. and Zhang, J. (2003). A zero public infrastructure vehicle based traffic information system. Proceedings of the 82nd TRB Annual Meeting, Washington, D.C.