INTEREST MANAGEMENT SCHEME AND PREDICTION MODEL IN INTELLIGENT TRANSPORTATION SYSTEMS

A Dissertation
Presented to
The Academic Faculty

by

Ying Li

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering, College of Computing

Georgia Institute of Technology
December 2012

INTEREST MANAGEMENT SCHEME AND PREDICTION MODEL IN INTELLIGENT TRANSPORTATION SYSTEMS

Approved by:

Dr. Richard Fujimoto, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Michael Hunter
School of Civil and Environmental Engineering
Georgia Institute of Technology

Dr. Guy Lebanon
School of Computational Science and Engineering
Georgia Institute of Technology

Dr. Christos Alexopoulos
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Richard Vuduc
School of Computational Science and Engineering
Georgia Institute of Technology

Date Approved: October 8, 2012

ACKNOWLEDGEMENTS

First, I feel really grateful to my advisor Dr. Richard Fujimoto, one of the pioneers in the modeling and simulation field. This work would not be possible without his guidance and continous support throughout my doctoral study. He always gave me the freedom to explore the topics in which I am interested and provided advisement and guidance to help me perform research on these topics. I have learned a great deal not only from his expertise but also from his attitude to science. I feel I am extremely fortunate to study under his supervision.

I would like to express my gratitude to Dr. Michael Hunter who has been very supportive of my work. I am lucky to work with Dr. Hunter who has provided suggestions to broaden my horizons on the topic of intelligent transportation systems. I would like to also thank Dr. Christos Alexopoulos and Dr. Randall Guensler who have given me insightful comments and suggestions during group discussions. I thank Dr. Guy Lebanon and Dr. Richard Vuduc for serving as my committee members and providing me with advice and feedback on my research.

I benefited from collaborations and discussions with students in the College of Computing and School of Civil and Environmental Engineering. These include Ya-Lin Huang, Hoe Kyoung Kim, Wonho Suh, Dwayne Henclewood and Alfred Park. I appreciate all the help from Carolyn Young, Michael Terrell and Lometa Mitchell.

Finally, I would like express my gratitude to my parents and my husband. This work would not be possible without their love and support all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Peer-to-peer systems eliminate the bottleneck caused by centralized entity management. Mobile peer-to-peer systems are especially challenging because their dynamic nature introduces new problems in areas such as disseminating information and predicting future behaviors. These two problems are very important in Dynamic Data Driven Applications Systems (DDDAS) that use online information to optimize system performance. One important aspect of DDDAS concerns the incorporation of data into an application. Distribution of data from sensors to mobile applications becomes an important problem in DDDAS. Data distribution management (DDM) controls the distribution of data among data producers and data consumers in a distributed application. Prediction of the future behavior of a complicated system is also an important part of a DDDAS deployment. Predictions can be used to dynamically adapt the operation of a complex system to steer the system toward more desirable states.

This thesis focuses on two important problems related to DDDAS: interest management (data distribution) and prediction models. In order to reduce communication overhead, we propose a new interest management mechanism for mobile peer-to-peer systems. This approach involves dividing the entire space into cells and using an efficient sorting algorithm to sort the regions in each cell. A mobile landmarking scheme is introduced to implement this sort-based scheme in mobile peer-to-peer systems. The design does not require a centralized server, but rather, every peer can become a mobile landmark node to take a server-like role to sort and match the regions. Experimental

results show that the scheme has better computational efficiency for both static and dynamic matching.

In order to improve communication efficiency, we present a travel time prediction model based on boosting, an important machine learning technique, and combine boosting and neural network models to increase prediction accuracy. We also explore the relationship between the accuracy of travel time prediction and the frequency of traffic data collection with the long term goal of minimizing bandwidth consumption. Several different sets of experiments are used to evaluate the effectiveness of this model. The results show that the boosting neural network model outperforms other predictors.

CHAPTER 1

INTRODUCTION

Much research to date has examined peer-to-peer architectures and applications in the context of fixed and wired network infrastructures. Recently developed wireless communication technologies and the new available capacities presented in mobile devices have enabled a novel peer-to-peer paradigm to emerge that focuses mainly on mobile and dynamic environments. This type of system is referred to as a mobile peer-to-peer system. Mobile peer-to-peer systems [26] are especially challenging relative to peer-to-peer systems built over wired infrastructures because their dynamic nature introduces new problems in areas such as disseminating information and prediction of the future behavior of the system.

In a complex system it is difficult to analyze and accurately predict future behaviors. Application simulations that can dynamically incorporate new data from historical statistics or on-line measurement can, in principle, result in more accurate analysis and prediction. These capabilities are central to the Dynamic Data Driven Applications Systems (DDDAS) paradigm [2]. DDDAS relies on the ability to incorporate additional data into an executing application as well as the ability to utilize the analysis and prediction capabilities of application simulations.

One important aspect of DDDAS concerns the incorporation of data into an executing application. Distribution of data from sensors to mobile applications is an important problem in DDDAS. Data distribution management (DDM) controls the distribution of data among data producers and data consumers. The DDM system must determine the consumers that are interested in receiving data generated by producers and deliver the corresponding message to these consumers.

Prediction of the future behavior of a complex system is an important component of a DDDAS deployment. Predictions are needed to help users and the DDDAS system itself to manage and optimize the system being managed. The predictions can be used to dynamically adapt the operation of a complex system to steer the system toward more desirable states.

One motivating application for DDDAS concerns infrastructure-less intelligent transportation systems. While most existing intelligent transportation system (ITS) deployments rely on a fixed communication and traffic management infrastructure, infrastructure-less systems offer the possibility to create ITS deployments at a much lower cost. This is one example application of a mobile peer-to-peer system. Vehicles with computing and communication capabilities communicate with each other without centralized entity management and share information throughout their movement. Interest management (data distribution) is essential for reducing communication overhead by filtering irrelevant messages in infrastructure-less intelligent transportation systems. In addition to interest management, travel time prediction is another important problem in intelligent transportation systems because such predictions can potentially enable drivers to make decisions and plan their routes to reduce congestion and delay.

The major elements of this thesis include: peer-to-peer architecture, Dynamic Data Driven Applications Systems (DDDAS) and distributed simulation. The High Level Architecture (HLA) provides a context for data distribution management. Travel time prediction models in intelligent transportation systems are also examined. Each of these is discussed next.

1.1 Peer-to-Peer Architecture

Peer-to-peer (P2P) architecture has been a topic of growing interest in recent years. Peer-to-peer applications that have been studied include file sharing [16], distributed computing [12] and P2P networked virtual environments [14, 15, 26].

It is instructive to contrast the peer-to-peer software architecture with the client-server architecture. The client-server architecture is a computing model that is usually implemented as a distributed application that partitions tasks or workloads between the providers of a resource or service, called the servers, and service requesters, called clients [8]. A server machine is a host that is running one or more server programs that serve the needs of the clients and share the resources with them. A client initiates communication sessions with a server and generates service requests to the server; the server awaits incoming requests, accepts and processes them, and then returns the results or information to the client.

Figure 1 depicts communications between a server and a client as well as between two clients in the client-server architecture. Client 1 first initiates a communication session with the server. The server receives and processes incoming requests. After processing the requests, the server replies to the client, typically providing requested information or confirmation of completion of the requested service. If two clients would like to exchange information, they must communicate through the centralized server. For example, in Figure 1, client 2 must first upload content to the server so it can be retrieved from the server by client 1.

Although client-server models are useful in many situations, there are several challenges with this type of architecture.

1) Scalability: If there are a large number of clients, a large number of requests from the clients may be generated, leading to overloading the servers.
2) Robustness: A server failure may lead to client requests that cannot be fulfilled. Additional mechanisms are required to address this problem.
3) Cost: Servers can be very expensive to install and maintain.

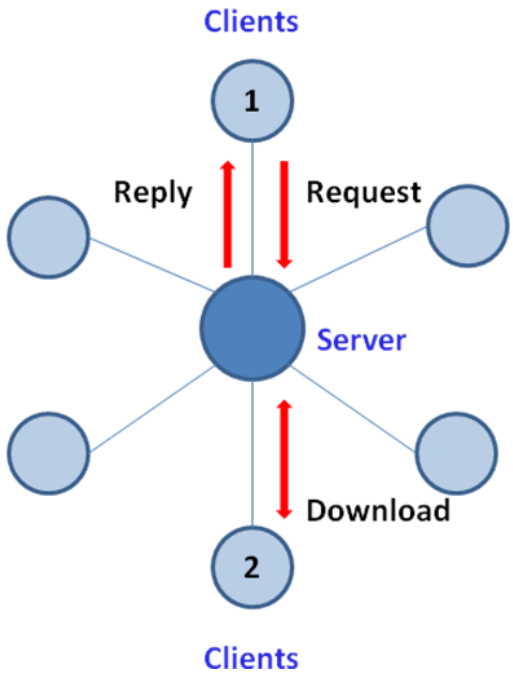


Figure 1. Communication Model in Client-Server Architecture

Peer-to-peer architectures offer the potential to address these concerns. Such architectures are now being utilized for many real-world applications such as distributed computing, content sharing and instant messaging.

The peer-to-peer model refers to an architecture where each computer in the network can act as a client or server for the other computers in the network, allowing shared access to files and peripherals without the need for a central server [4]. In peer-to-peer architectures all of the tasks are distributed among the peers. Each peer has the same privileges and capabilities and can initiate a communication session. Peers are both suppliers and consumers of resources, in contrast to the traditional client–server model where only servers supply, and clients consume [4].

1.1.1 Peer-to-Peer Categories

Peer-to-peer architectures can be categorized as being structured or unstructured. Unstructured peer-to-peer networks can be further classified into three different

architectures: centralized architecture, decentralized architecture and hybrid architecture (semi-centralized architecture). Peers communicate directly with each other in all of these architectures. Differences among them concern the logical network topologies and how information is located within the network [27]. Figure 2 depicts a hierarchy of peer-to-peer models.

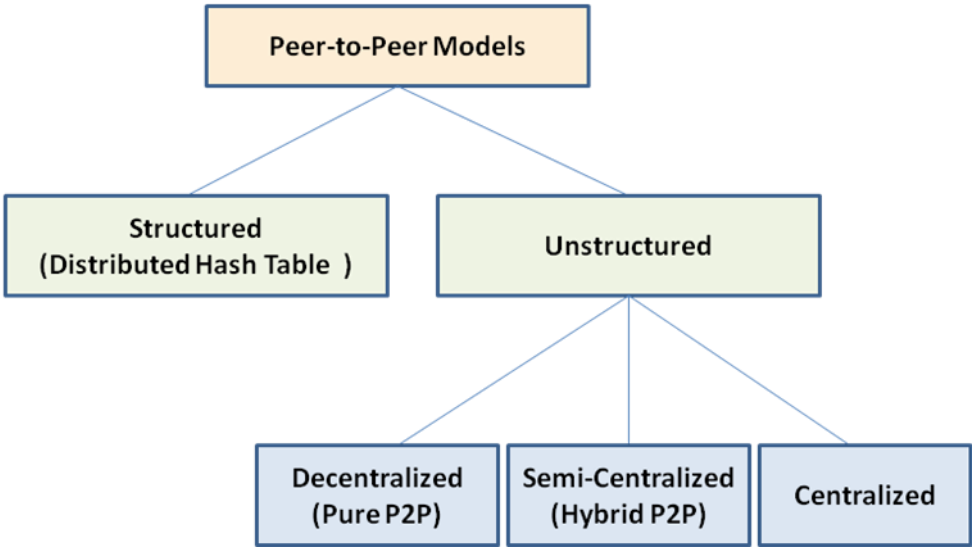


Figure 2. Peer-to-Peer Categories

In structured peer-to-peer models, peers are organized following specific criteria and algorithms, which lead to overlays with specific topologies and properties [4]. They typically use distributed hash table-based (DHT) indexing, which can ensure that any node can efficiently route a search to some peer that has the desired file.

Distributed hash tables are a class of decentralized distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in the DHT, and any participating node can efficiently retrieve the value associated with a given key [5]. Figure 3 shows the architecture of distributed hash tables. In a file sharing system the value corresponds to the content of the file, and the key to a hash of the filename. (key, value) pairs are stored in P2P nodes. Any participating node can retrieve the value

associated with a given key by using a peer-to-peer overlay network to map keys to nodes. There are two important operations: insertion will add a (key, value) pair into the peer-to-peer overlay network while lookup returns the ID of the node that stores the (key, value) pair.

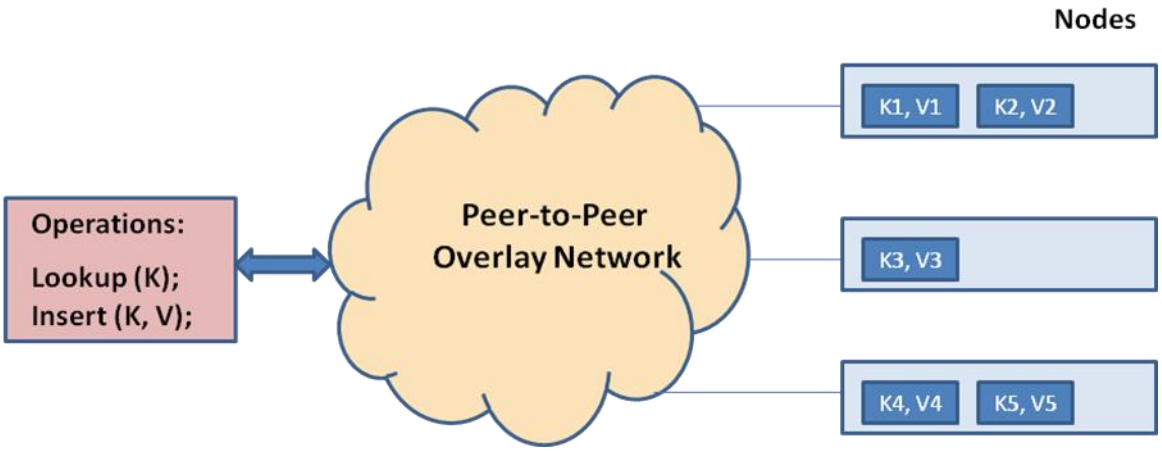


Figure 3. Distributed Hash Tables

In unstructured peer-to-peer architectures, peers connect with each other in an ad-hoc fashion without imposing any structure on the overlay networks. Unstructured peer-to-peer systems have different degrees of centralization, although ideally, systems should have no centralized management. In decentralized systems (pure peer-to-peer systems), all the peers are equipotent and there is absolutely no centralization. Semi-centralized systems (hybrid peer-to-peer systems) allow Super-Peers to exist, which have some special infrastructure function. In centralized peer-to-peer systems, a central server is used as a centralized database to locate resources (indexing function).

The decentralized architecture, also called a pure peer-to-peer architecture, only consists of equipotent peers. Searches in pure peer-to-peer architecture are done by forwarding queries from node to node, using broadcasting to send out the search request (e.g. in Gnutella 0.4), or by a more intelligent routing method (e.g. Freenet) which builds a heuristic key-based routing table, but the heuristic method does not guarantee the file will be found after search. In both approaches, if some node has the requested file, it

replies to the original querying peer via the same path as the original query route [27]. For example, in Figure 4, the queries from peer 1 are forwarded from node to node. Peers 2 and 4 reply to peer 1 via the same path as the original query route. Peer 1 then communicates with peer 2 and downloads the file from peer 2.

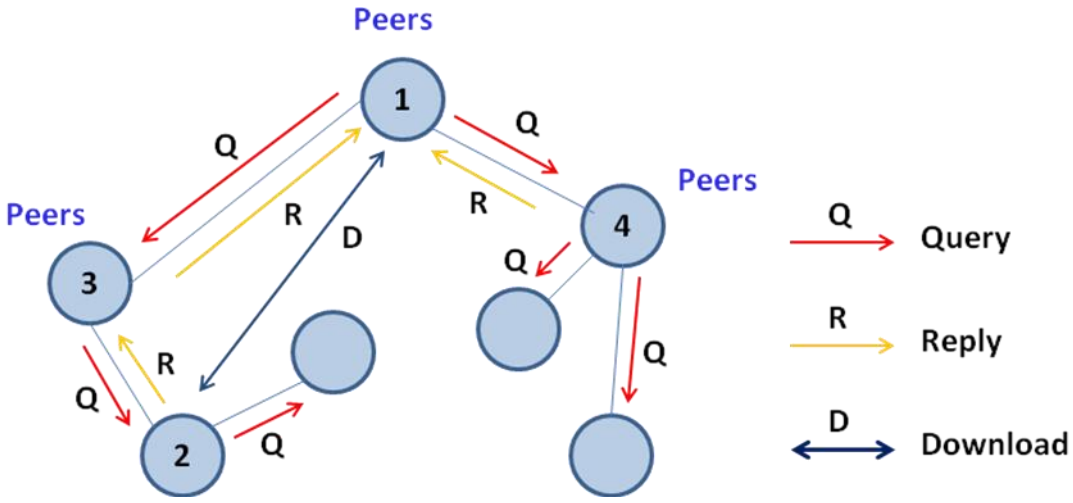


Figure 4. Decentralized Architecture (Pure Peer-to-Peer Model)

The centralized peer-to-peer architecture allows a centralized server, a Super-Peer (SP), to be used as a centralized database to store the information about all the files in the entire system. Ordinary peers upload information about their files to this server and the server holds the information for future queries. A node will send query request to the centralized server when it searches for files; the server checks its internal database, locates the peers that have the requested files and replies to the querying node with this information. The querying node then communicates directly with one peer in the reply without the SP involvement. For example, in Figure 5, peer 1 queries the central server and then the server replies indicating that peer 2 has the requested resource. After that, peer 1 and peer 2 communicate with each other directly without the involvement of the server.

The centralized peer-to-peer architecture is different from the client-server approach. In the centralized peer-to-peer architecture, the centralized server operates as a centralized database to locate resources, and peers communicate directly with each other. In the traditional client-server models, peers do not communicate with each other at all, but only with the centralized server.

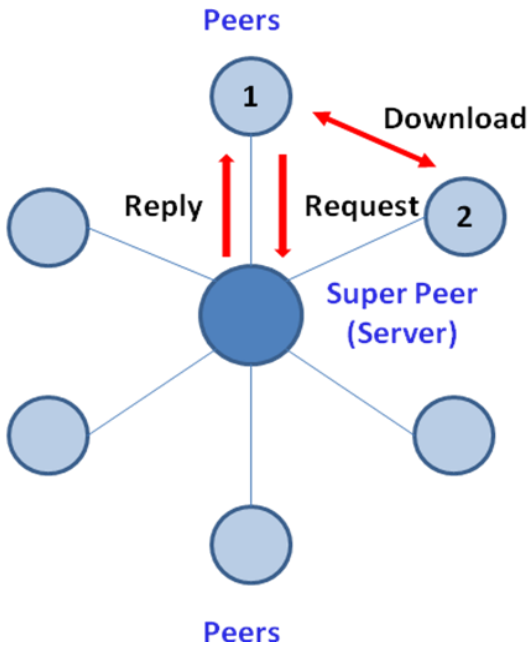


Figure 5. Centralized Peer-to-Peer Architecture

The semi-centralized architecture (hybrid architecture) is a combination of the centralized and the pure peer-to-peer architectures. The hybrid models have several Super-Peers and these Super-Peers form a pure peer-to-peer architecture. Ordinary peers connect to Super-Peers as in the centralized architecture. Each Super-Peer has a database to record the information about the files held in the ordinary peers connected to it. Ordinary peers initiate queries and queries are forwarded to the Super-Peer connected with the ordinary peers. Super-Peers then query each other if any of their ordinary peers has the searched files. If there is a match, all the future communication is performed among ordinary peers without the involvement of Super-Peers [27].

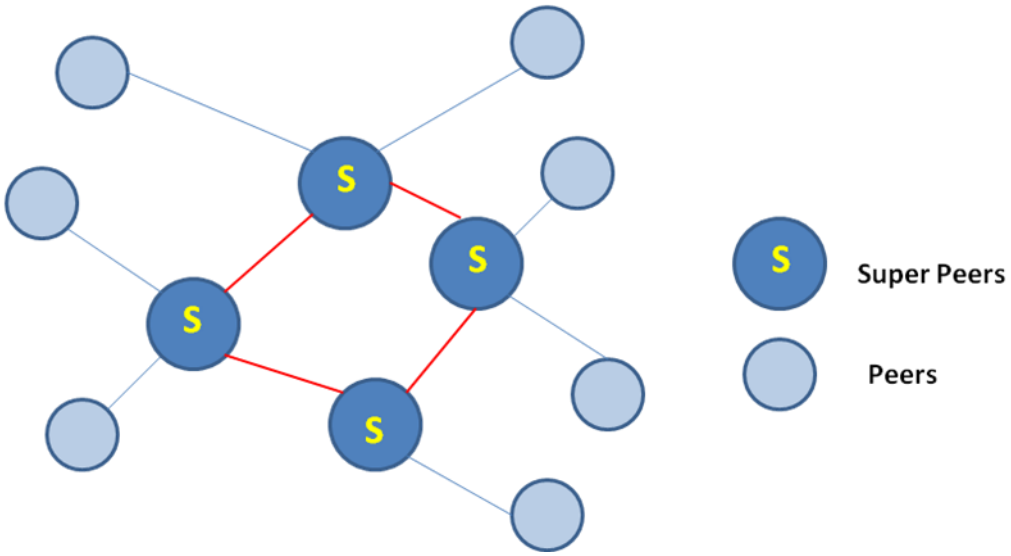


Figure 6. Semi-Centralized Architecture (Hybrid Peer-to-Peer Model)

No matter which type of architecture is used, the computation and communication overhead is shared among joining peers in the peer-to-peer architecture, so the bottleneck due to centralized entity management is removed. This is the main advantage of the peer-to-peer architecture relative to the client-server architecture.

1.2 Dynamic Data Driven Application Systems

It is difficult to accurately analyze and predict the behavior of complex systems. Application simulations that can dynamically incorporate new data, no matter whether the data is from historical statistics or from on-line measurements of the actual systems, can result in more accurate analysis and more accurate prediction. These capabilities are utilized with the Dynamic Data Driven Applications Systems (DDDAS) paradigm [2]. DDDAS exploits the ability to incorporate additional data into an executing application as well as the ability to augment the analysis and prediction capabilities of the application simulations.

Dynamic Data Driven Applications Systems have been widely studied and applied to various science and engineering disciplines. A typical application is to enhance the

simulation by additional data incorporated into the computation. For example, for crisis management applications such as fire propagation in buildings [28], data-driven simulation of fire propagation dynamics can be modeled since the situation may evolve with new unforeseen events arising. These enhanced modeling and simulation approaches can provide more accurate information to firefighting personnel and an effective response [2]. Other dynamic data driven applications include prediction of regional scale weather phenomena [31], modeling of subsurface and ambient atmosphere pollutants [29, 30], optimization of surface transportation systems [32, 33], management of semiconductor manufacturing systems [34], and recognization of image-based human stress [35].

1.3 Distributed Simulation

A computer simulation is a computer program that attempts to model the behavior of a physical system over time. Simulation is a multi-disciplinary field, with research dispersed across multiple fields of study. Distributed simulation refers to technologies that enable a simulation program to execute on multiple computers interconnected by a communication network.

Distributed simulation provides multiple potential benefits:

1) Integrating geographically distributed simulators: This capability allows simulators to be executed on computers at distinct geographical locations to create virtual worlds with participants located at different sites.
2) Integrating proprietary simulators (e.g., commercial-off-the-shelf tools): This may require the simulation computation to be distributed across multiple computers (or virtual machines) if the simulators require different operating systems.
3) Composing multiple disparate models: It is often more cost effective to interconnect existing simulators, each executing on a different computer, rather than to create an entirely new simulator.

1.3.1 Distributed Interactive Simulation

Distributed interactive simulation (DIS) is a set of standards for creating distributed virtual environments. A principal objective of DIS (and subsequently the High Level Architecture effort) is to enable interoperability among separately developed simulators.

A DIS exercise may include the following different types of simulators [1]:

1) human-in-the-loop systems (also called virtual simulators, such as tank simulators)
2) computation only elements (also called constructive simulations, such as wargame simulations)
3) live elements (such as instrumented missiles)

One principle feature of DIS is that the simulation nodes are autonomous. This simplifies development since developers of one simulator can focus the design and implementation of their own simulator without being overly concerned with the detailed information of other simulators. Autonomy among simulation nodes should be maintained in two ways. One is concerned with time management while the other with data distribution management. Time management refers to the management services to coordinate the advancement of simulation time, while data distribution management controls the distribution of state updates and interactions among simulators.

1) No synchronization among simulators is used to advance simulation time in DIS. Each simulator in DIS advances simulation time autonomously in synchrony with wallclock time.
2) There is no need to specify the recipients of messages in DIS. The simplest approach is to broadcast each message to all the simulators; then each simulator decides for itself whether the message is relevant to itself. Other approaches include the publication and subscription services used in the High Level Architecture, as will be discussed in detail later.

In DIS, simulation nodes only transmit information concerning state changes. Information concerning objects that do not change need not be transmitted to other simulators. For example, in transportation systems, the current location, speed and direction in which a vehicle is moving should be transmitted over the network. However, it will incur an excessive amount of communication overhead if an entity sends each update to these variables. Rather, one can have each entity send state update information periodically or using dead reckoning techniques where an entity sends the information to other simulation nodes only if its position deviates from an extrapolation of the last update by some threshold. Dead reckoning algorithms can reduce the amount of required communication significantly.

1.3.2 High Level Architecture

The High Level Architecture (HLA) is a general purpose architecture for distributed computer simulation systems. Like DIS, the principal goal of the High Level Architecture is to support interoperability and reuse of simulations. The purpose behind the development of HLA is described as follows on the US Defense Modeling and Simulation Website (DMSO) [3]:

> The Department of Defense (DoD) Modeling and Simulation (M&S) Master Plan calls for the establishment of a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations including C4I systems, and to facilitate the reuse of simulation components. The HLA, developed in response to the Master Plan, provides a systematic and consistent basis for addressing simulation system design and implementation issues, facilitates interoperability and reuse through a set of common rules, and furnishes a framework for making policy decisions.

The HLA designers had five principle goals [6]:

1) It should be possible to decompose a large simulation problem into smaller parts; smaller parts are easier to define, build correctly, and verify.
2) It should be possible to combine the resulting smaller simulations into a larger simulation system.
3) It should be possible to combine the smaller simulations with other, perhaps unanticipated simulations to form a new system.
4) Those functions that are generic to component-based simulation systems should be separated from specific simulations. The resulting generic infrastructure should be reusable from one simulation system to the next.
5) The interface between simulations and generic infrastructure should insulate the simulations from changes in technology used to implement the infrastructure, and insulate the infrastructure from technology in the simulations.

In HLA terminology, each individual simulator is called a federate and the combined distributed simulation system is known as a federation. The High Level Architecture consists of three components:

1) *Rules* that define the underlying design principles in the HLA.
2) *Object Model Template (OMT)* that specifies a common format to record the information in the object models including objects, attributes and interactions.
3) *Interface Specification* that defines how HLA compliant simulators interact with the Run-Time Infrastructure (RTI). The RTI is analogous to a distributed operating system providing distributed simulation services.

The HLA rules are technical principles that must be followed to make federates interact with each other properly during a federation execution. They describe the responsibilities of federates (simulators of different type, which will be illustrated later)

and federations (distributed simulation system consisting of multiple federates). For federates, the rules specify the requirement of a Simulation Object Model (SOM), time management requirement, and functionalities of updating or reflecting attributes as well as transferring or accepting attributes ownership. For federations, the rules specify the requirement of a Federation Object Model (FOM), functionalities and constraints related to RTI and Interface Specification.

HLA requires each federate and federation to have an object model describing the objects used in the simulations as well as attributes and interactions. Object models in the HLA are documented using a set of tables that are collectively referred to as the object model template (OMT) [1]. The HLA OMT specifies how to record the information in the object models, but it does not define the specific data (such as vehicles, positions and so on). For example, each federate must have a SOM which has a tabular format with an object class structure table, an attribute table and an interaction class structure table. The object class structure table specifies the class hierarchy of objects, the attribute table provides information about object attributes while the interaction class structure table describes the object actions and interactions. Each federation must have a FOM which has the same structure as the SOM and identifies the attributes and interactions supported by the federation. All participating federates must use the same FOM [36].

In HLA, federates interact with an RTI to support efficient information exchange among each other. Federates here may be software simulations (such as vehicle simulators), live components (such as instrumented tanks), or passive data viewers. The HLA interface specification is the specification of the interface between federates and the RTI (see Figure 7).

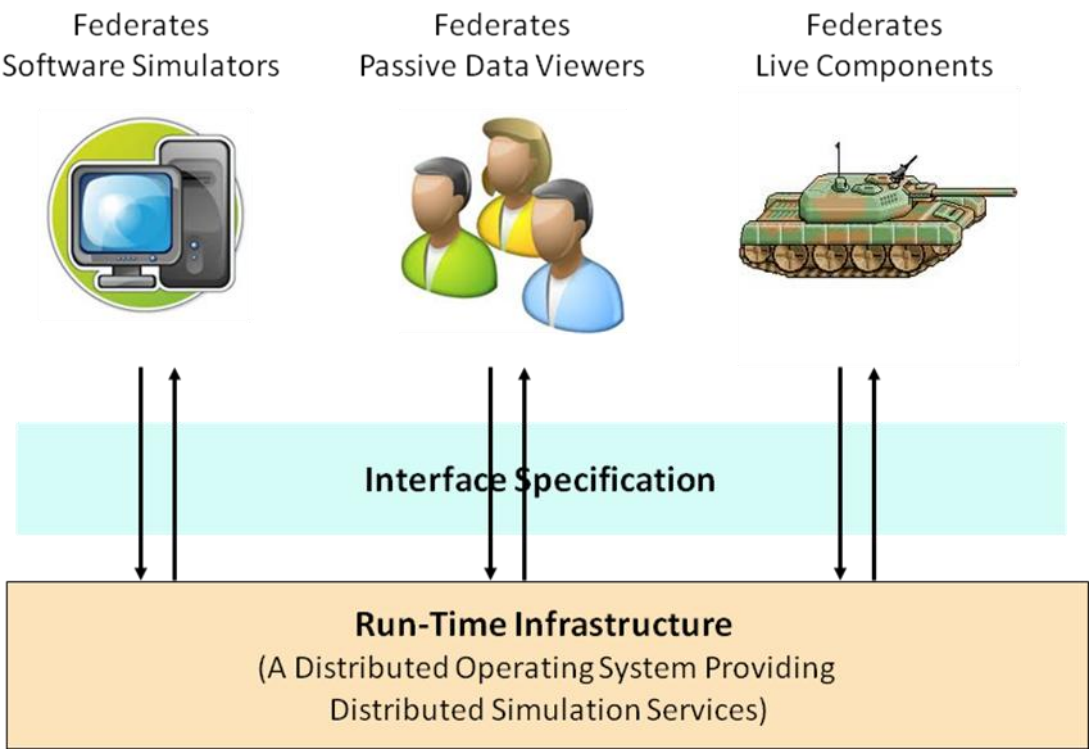


Figure 7. Components of an HLA Federation

In the HLA, the state variables for the federation are stored within federates rather than the RTI, and the RTI is a general software component that is applicable to any federation. The interface specification defines a set of services provided by simulations or by the RTI during a federation execution. There are six categories of services [1]:

1) *Federation Management:* These services create and delete federation executions, and allow federates to join or resign from existing federations.
2) *Declaration Management:* These services allow federates to establish their intent to publish object attributes and interactions, and to subscribe to updates and interactions produced by other federates.
3) *Object Management:* These services allow federates to create and delete object instances, and to update attributes (send messages) and receive updates to attributes (receive messages) produced by other federates.

4) *Ownership Management:* These services enable the transfer of ownership of object attributes during the federation execution.
5) *Time management:* These services control the advancement of simulation time within each federate, and prevent federates from receiving messages in their simulated past.
6) *Data distribution management:* These services control the distribution of information among federates so that federates receive all of the information relevant to them and (ideally) no other information.

These services can be illustrated by a typical federation execution, which includes four major steps: initializing the federation; declaring objects of common interest among federates; exchanging information and terminating execution. The process is shown in the Figure 8.

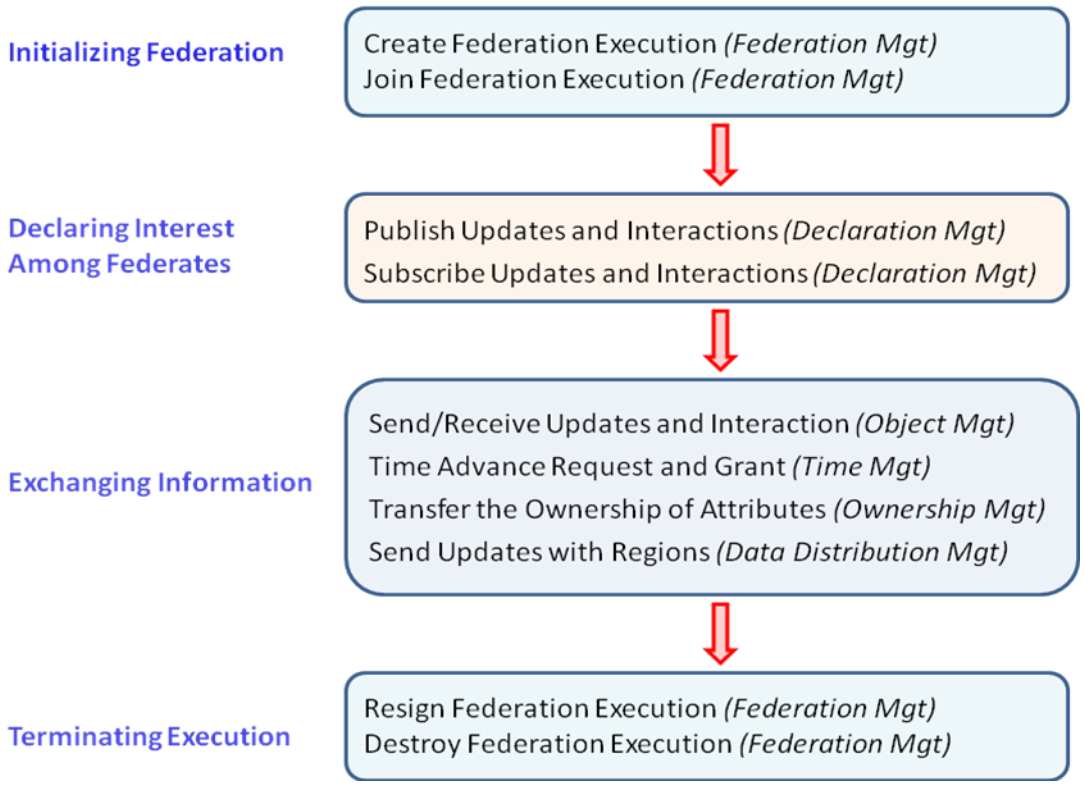


Figure 8. Services in a Typical Federation Execution

1.4 Data Distribution in the High Level Architecture

Data distribution management (DDM) controls the distribution of information among information producers and information consumers in a distributed simulation application. For the information produced, the DDM system must determine who is interested in the information and deliver the message containing the information to the right receivers (information consumers). To be able to do this, the DDM system must include a way to describe the information produced by information producers as well as a way for consumers to express the type of information they wish to receive.

In general, the DDM framework includes a *name space* and a language to specify *expressions* which are subsets of the name space [1]. The name space is a common vocabulary used to describe the produced information and to express interests in information. An *interest expression* is a subset of the name space; it is used to specify what information a consumer is interested in receiving. A *description expression* is also a subset of the name space; it is used to describe the information generated by a producer. When the description expression of a message containing some information overlaps with an interest expression of an information consumer, the message should be sent to that consumer.

More formally, the basic concepts in DDM framework are defined as follows:

Name space: The name space is a set of all possible values of interest expressions and description expressions. The name space is a set of tuples (V_1, V_2 , ... , V_N) where each V_i is a value of some basic type, or another tuple. For example, in a school bus service system, school buses may be interested in the number of passengers of other buses in order to adjust future schedules. The name space could be defined as a tuple (V_1, V_2), where V_1 is an enumerated type specifying the different types of school buses (for example, red route bus, blue route bus and green route bus, each of which services a different route in the campus), and V_2 is an integer type specifying the number of

passenger in a school bus.

Interest Expressions: An interest expression is a subset of the name space. It is used to specify what information is to be received. For example, the interest expression for all red route buses with more than X passengers beyond the maximum load X_0 is all tuples (red route buses, X) such that $X > X_0$.

Description Expressions: A description expression is also a subset of the name space. It is associated with each message and used to describe the contents of the message. For example, a simulator modeling a green route bus with 30 passengers may generate a message with description expression (green route bus, 30).

The name space, interest expressions, and description expressions are important elements in DDM. DDM is necessary for interconnecting applications in distributed simulations. For example, in HLA, two types of data distribution services are provided: static data distribution (also called class-based data distribution) and dynamic data distribution (also called value-based data distribution).

1.4.1 Static Data Distribution in HLA

In the HLA the federation must define a FOM that specifies a hierarchy indicating object classes and their attributes. For example, in the above transportation simulation example, one can define a class called *school bus*, with subclasses called *red route bus*, *blue route bus* and *green route bus*. These class definitions are used for class-based data distribution (also called static data distribution). Specifically, each federate should subscribe to attributes of a specific class in the class hierarchy tree. For example, a federate may subscribe to the *passenger number* attribute of the *school bus* class and it will be notified whenever a school bus sends updates of its passenger number attribute.

Figure 9 shows a sample class hierarchy tree for this school bus simulation system. Besides defining its own attributes, a class inherits all the attributes from its parent class. For example, an attribute *passenger number* can be defined in the *school bus* class, and inherited by all three subclasses: *red route bus*, *blue route bus* and *green route bus*. Thus the name space consists of the tuples *(school bus, passenger number), (red route bus, passenger number), (blue route bus, passenger number)* and *(green route bus, passenger number).*

There are at least two approaches to implement class-based data distribution. The first approach is based on the following idea: when a federate subscribes to an attribute of a class, it automatically subscribes to that attribute inherited by the subclasses in the subtree rooted at that class [1]. For example, in Figure 9, a subscription to *(school bus, passenger number)* becomes an interest expression enumerating a set of tuples *(school bus, passenger number)*, *(red route bus, passenger number)*, *(blue route bus, passenger number)*, and *(green route bus, passenger number)*. The description expression for an update to an attribute of a class includes only one tuple, such as *(red route bus, passenger number)* when the passenger number of a red route bus object is updated. Federates subscribed to *(school bus, passenger number)* and *(red route bus, passenger number)* will receive updates associated with this description expression, while those only subscribed to *(blue route bus, passenger number)* and *(green route bus, passenger number)* will not be notified of the updates.

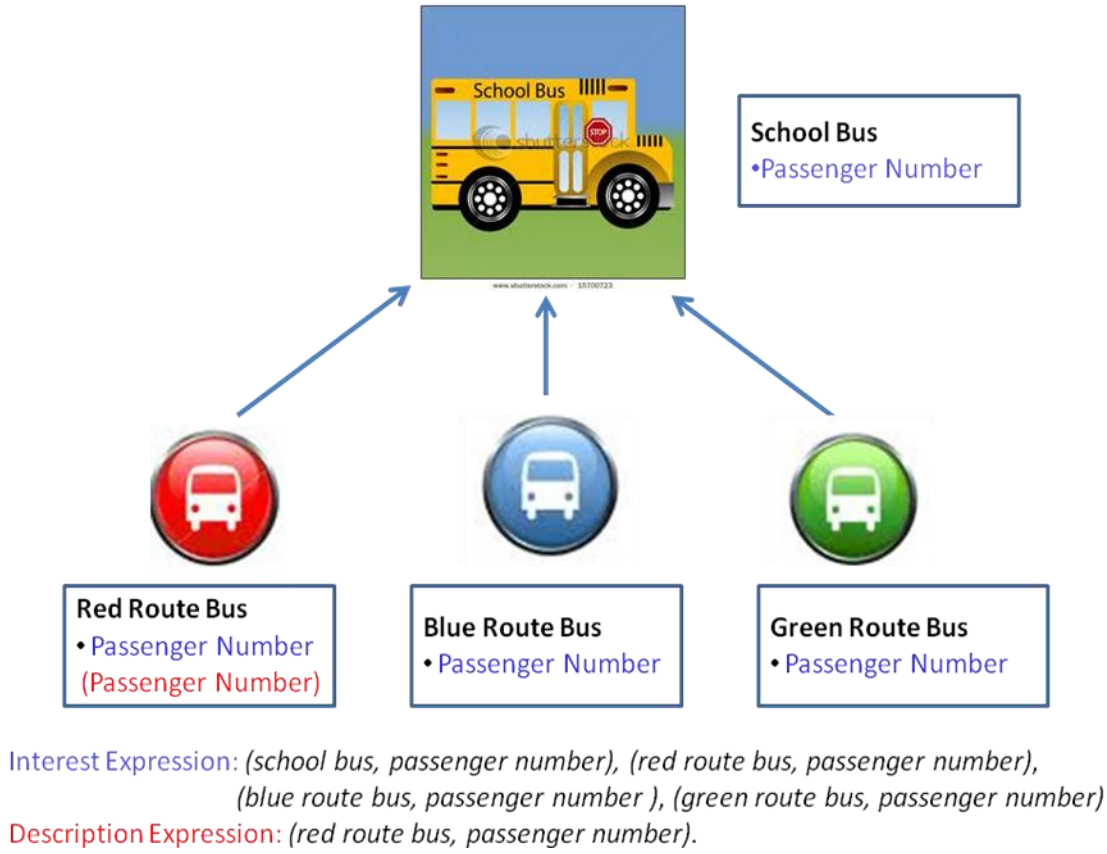


Figure 9. Class-Based Data Distribution (Approach I)

Another approach defines an interest expression containing only a single tuple, while a description expression for an attribute of a class contains multiple tuples including the classes along the path from this class up the class hierarchy tree to the class where this attribute was originally defined [1]. For example, in Figure 10, an interest expression *(school bus, passenger number)* indicates the interest in the passenger number attribute of class school bus. An update to the passenger number attribute of a red route bus object contains the description expression *(red route bus, passenger number)* and *(school bus, passenger number)*. This approach will yield the same result as the first approach: federates subscribed to *(school bus, passenger number)* and *(red route bus, passenger number)* will be notified of the updates, but those only subscribed to *(blue route bus, passenger number)* and *(green route bus, passenger number)* will not receive

the updates.

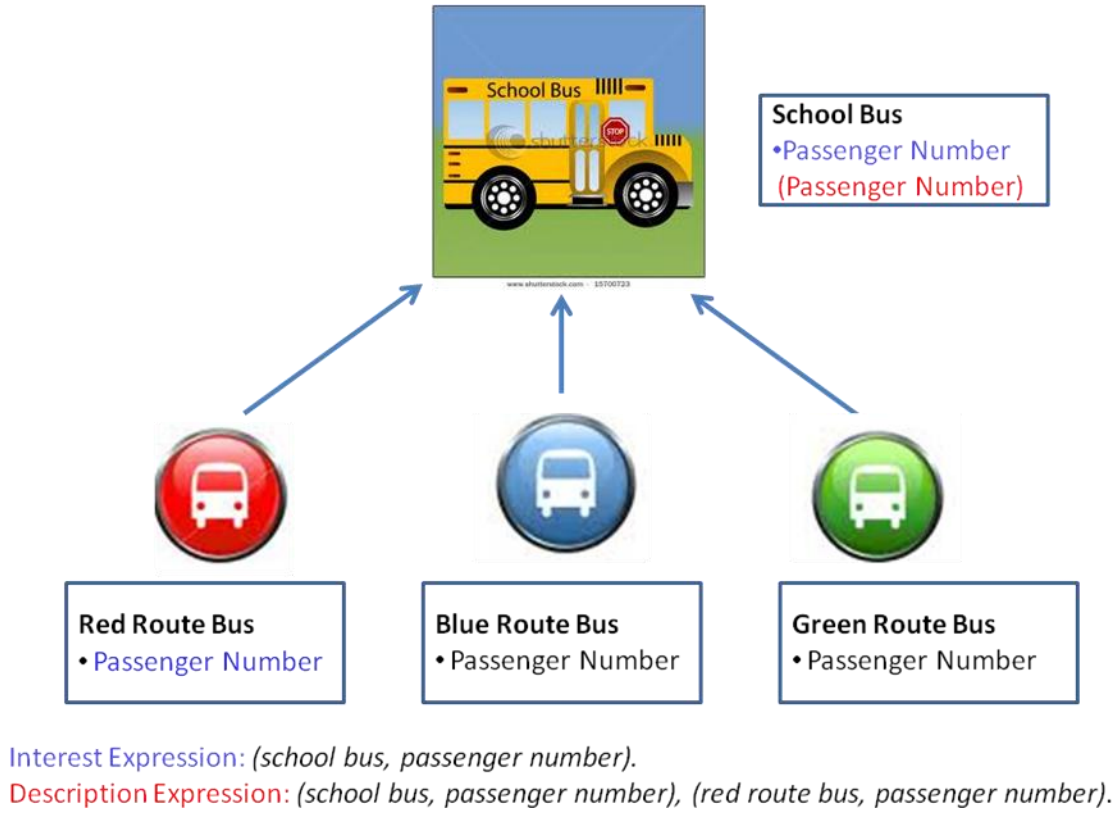


Figure 10. Class-Based Data Distribution (Approach II)

1.4.2 Dynamic Data Distribution in HLA

The dynamic data distribution is also called value-based data distribution,which is based on an n-dimensional coordinate system called a *routing space*. For example, a two-dimensional routing space might be used to represent the geographical area covered by the virtual environment. Interest and description expressions in the HLA define areas called *regions* of a routing space. Interest expressions are referred to as *subscription regions*, and description expressions are referred to as *update regions*. For example, the routing space in Figure 11 includes one update region U and two subscription regions Sl and S2.

The basic rule about value-based data distribution is as follows: If the update region associated with a message overlaps with a federate's subscription region, the message should be routed to the subscribing federate [1]. For example, suppose that the routing space in Figure 11 corresponds to the geographic area of a virtual environment that includes moving vehicles of an intelligent transportation system. Assume that the update region U indicates a vehicle inside this region that can send updates to this area. S1 and S2 are the subscription regions created by federates F1 and F2, respectively. Both F1 and F2 model moving vehicles. F1 is interested in vehicles' updates within S1 while F2 is interested in vehicles' updates within S2. If the vehicle inside U moves to a new position, a message about the updated position will be sent to F1 because its subscription region Sl overlaps with U, while F2 will not receive the updates since its subscription region S2 does not have an overlapping area with U.

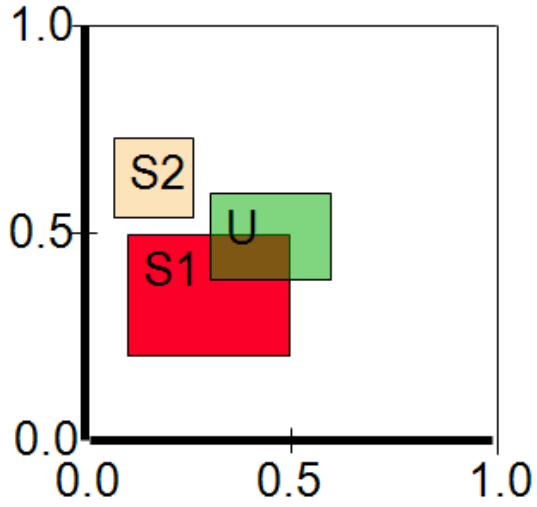


Figure 11. Subscription Region and Update Region in Routing Space

There are multiple approaches to implement dynamic data distribution, such as the grid-based approach, region-based approach, hybrid approach and sort-based approach. Each approach will be described in detail in Chapter 2.

1.5 Intelligent Transportation Systems

A motivating application of the research described here is infrastructure-less intelligent transportation systems [23]. They are an example application of mobile peer-to-peer systems.

Intelligent transportation systems (ITS) utilize computational and communications technology in the transportation infrastructure and vehicles with the goal of improving mobility, enhancing safety, reducing pollution, and reducing travel times. Interest in ITS is motivated by the problems caused by traffic congestion and emergence of new information technology for simulation, communications networks as well as real-time control. The current trend in ITS involves making vehicles and roadways "smarter" through advanced technologies.

Intelligent transportation systems include a wide and growing suite of technologies and applications. ITS applications can be grouped within five main categories [7]:

1) Advanced Traveler Information Systems provide drivers with real-time information, such as transit routes and schedules, navigation directions, and information about delays due to congestion, accidents, weather conditions, or road repair work.
2) Advanced Transportation Management Systems include traffic control devices, such as traffic signals, ramp meters, variable message signs, and traffic operations centers.
3) ITS-Enabled Transportation Pricing Systems include systems such as electronic toll collection (ETC), congestion pricing, fee-based express (HOT) lanes, and vehicle miles traveled (VMT) usage-based fee systems.
4) Advanced Public Transportation Systems, for example, allow trains and buses to report their position so passengers can be informed of their real-time status (arrival and departure information).

5) Fully integrated intelligent transportation systems, such as vehicle-to-infrastructure (VII) and vehicle-to-vehicle (V2V) integration, enable communication among assets in the transportation system, for example, from vehicles to roadside sensors, traffic lights, and other vehicles.

In this thesis work, we focus on the first and fifth categories of ITS applications. One goal is to provide the drivers real-time information such as travel time prediction to help them make decisions, e.g., to plan routes. We also explore communication in infrastructure-less intelligent transportation systems that utilize vehicle-to-vehicle (V2V) communication.

ITS offers five potential classes of benefits [7]:

1) Improving operational performance, particularly by reducing congestion
2) Enhancing mobility and convenience
3) Increasing safety
4) Delivering environmental benefits
5) Boosting productivity and expanding economic and employment growth

The contribution of this thesis work is centered on the first two benefits. The details will be illustrated in later chapters.

1.6 Research Problem and Contribution

This thesis will focus on two important problems in intelligent transportation systems: interest management and travel time prediction. In order to reduce the communication overhead in intelligent transportation systems, we propose a new interest management mechanism for peer-to-peer mobile systems. This approach involves dividing the entire space into cells and using an efficient sorting algorithm to sort the regions in each cell. A mobile landmarking scheme is introduced to implement this sort-

based scheme in infrastructure-less transportation systems. The design does not require a centralized server, but rather, every peer can become a mobile landmark node to take a server-like role to sort and match the regions.

In order to improve the communication efficiency in intelligent transportation systems, we present a prediction model based on boosting, an important machine learning technique, and combine boosting and neural network models to increase prediction accuracy. We also explore the relationship between the accuracy of travel time prediction and the frequency of traffic data collection with the long term goal of minimizing bandwidth consumption. The principal contributions of the thesis can be summarized as follows:

1. Propose a new interest management scheme with dynamic sorting. The proposed approach is well-suited for dynamic region modifications. If region changes occur, the scheme does not need to resort to the list of projections and repeat the entire matching process. Rather, it requires sorting and matching only for a subset of the regions.

2. Propose mechanism to improve computation efficiency for non-uniform distribution case. Random sampling is first used to determine the boundaries of the bucket sort. And then the rest of the data is distributed into buckets with the boundaries obtained in the first phase. During the runtime, merging and splitting buckets are performed to maintain equi-depth buckets.

3. Introduce and evaluate mobile landmarking as a means to implement interest management in mobile peer-to-peer systems. The design is distributed and does not utilize fixed servers. Rather, each peer can become a mobile landmark node to assume a server-like role to sort and match regions.

4. Propose a travel time prediction model based on boosting. The boosting approach is used as a method for travel time prediction in conjunction with neural network models to help capture characteristics of traffic and increase prediction accuracy.

5. Propose an approach to compute the lower bound of the data collection frequency. We examine the relationship between data collection frequency and travel time prediction accuracy, and propose an approach to compute the lower bound of the collection frequency while maintaining high prediction accuracy; to our knowledge, this is the first time Quality of Service is included as a factor in travel time prediction.

6. Demonstrate the influence of various factors on travel time prediction accuracy. We explore the influence of various factors on prediction accuracy such as monitoring interval times for historical data and the number of iterations used by the boosting algorithm.

1.7 Roadmap to This Document

The remainder of the thesis is organized as follows. Chapter 2 reviews the interest management schemes in the literature and presents an effective interest management scheme for mobile peer-to-peer systems. Chapter 3 reviews traditional and modern travel time prediction models and details a new online prediction model based on boosting. In chapter 4, we propose future research directions and propose an approach for exploiting parallel processing for interest management.

CHAPTER 2

INTERST MANAGEMENT SCHEME

2.1 Introduction

In peer-to-peer systems, the computation and communication overhead is shared among peers, removing the bottleneck caused by centralized entity management. Among peer-to-peer systems, mobile peer-to-peer systems [26] are especially challenging. One example application is infrastructure-less intelligent transportation systems [23]. Vehicles with computing and communication capabilities communicate with each other without centralized entity management and share information during movement. Broadcasting the data will consume far too many network resources. Message filtering schemes, or interest management schemes, must be used to effectively distribute data among peers.

The objective of interest management is to have all peers only receive data that are of interest to them. The data distribution management services of the High Level Architecture are based on identifying overlaps among pairs of subscription and update regions as explained in Chapter 1, and form the basis for the work described here. Implementation approaches include grid-based mechanisms (cell-based mechanisms) [11, 18] and region-based mechanisms (area-based mechanisms) [13, 25]. The grid-based algorithms divide the entire space into a grid of cells. An update region and a subscription region may overlap if they share at least one common grid cell. The overlapping information obtained by grid-based algorithms is not exact and unnecessary communication may be introduced. Region-based algorithms use a brute force approach. They compare all pairs of regions to determine overlaps. The computational process is straightforward but may incur a significant amount of computational overhead. The

hybrid approach [20] improves performance by exploiting the advantages of these two approaches. It first uses the grid-based approach to map all the regions to the grid cells. Then the region-based approach is used to determine the exact matching between update and subscription regions within each cell.

Sort-based algorithms [17, 21] have been used to compute the intersection between update and subscription regions. They project the regions on each dimension and sort the projections in order to determine overlaps. After obtaining the overlap information on each dimension, the matching between update regions and subscription regions can be computed because two regions overlap if the projections of the two regions intersect along each dimension. However, current sort-based algorithms have certain drawbacks. First, some algorithms statically process all regions in one round and must resort all regions again if any region changes, which can be very time consuming and does not scale. Second, current sort-based algorithms are based on comparison sorts, such as heapsort or insertion sort, whose computational complexity is O($n$log$n$). Third, current sort-based algorithms are all run in a centralized fashion where information concerning update and subscription regions are stored. This makes these algorithms poorly suited for peer-to-peer systems.

To address these drawbacks, we propose an interest management scheme for mobile peer-to-peer systems. Our interest management scheme has three advantages over other sort-based algorithms: 1) The algorithm is well suited for dynamic scenarios where region modifications are relatively frequent. The process of sorting and matching in dynamic scenarios is called dynamic sorting and matching. If region modifications occur, the algorithm does not need to resort the list of projections and conduct the entire matching again. It can do the dynamic sorting and matching without processing all of the regions. 2) Our scheme is based on bucket sort. Bucket sort is not a type of comparison sort and the average running time will be linear if not many projections fall into the same bucket. 3) We introduce the mobile landmarking design to implement this sort-based

scheme in mobile peer-to-peer systems. The design is distributed and does not utilize fixed servers. Rather, each peer can become a mobile landmark node to take the server-like role to sort and match regions.

In this chapter, we will describe major existing interest management schemes. We then present our interest management mechanism for mobile peer-to-peer systems. In this part, we illustrate our sorting and matching algorithm first, followed by the mobile landmarking scheme for implementing the sorting and matching algorithm in mobile peer-to-peer systems. Experiments and results are presented in later sections, which indicate that this approach yields better performance than several alternate interest management schemes.

2.2 Related Work

There are several major existing interest management schemes. These can be categorized as region-based approaches, grid-based approaches, hybrid approaches and sort-based approaches.

2.2.1 Region-based Approaches

The region-based or area-based scheme [13, 25] checks each update region (the region to which a node can send messages) with each subscription region (the region in which a node is interested) to derive the exact overlapping information. Figure 12 shows two update regions (U1 and U2) and two subscription regions (S1 and S2). In this example, U1 and S1 overlap and the updates from the object associated with U1 will be routed to the object of S1. However, U1 and S2 do not overlap and thus U1's updates will not be routed to S2. U2 overlap with both S1 and S2, and the updates from the object of U2 will be routed to both S1 and S2.

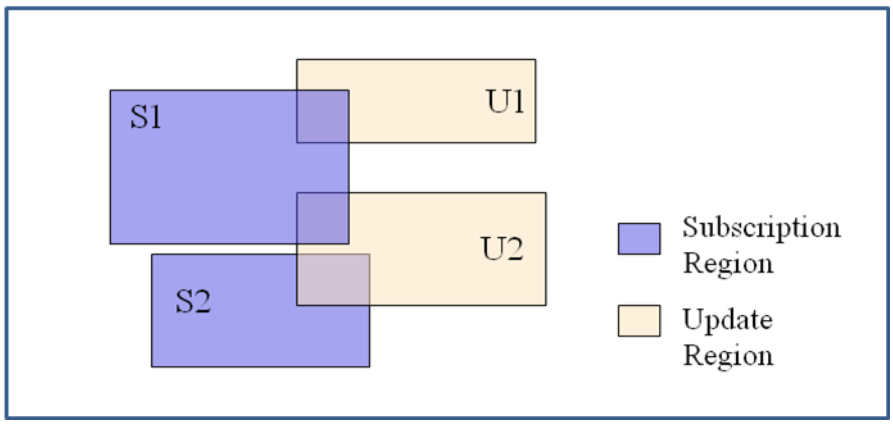


Figure 12. Region-based Approach

The main advantage of the region-based approach is its simplicity of implementation, since the algorithm is straightforward. The main problem of this mechanism is the high computational overhead and poor scalability. Since the matching requires each update region to be compared with all the subscription regions, the complexity is still $O(N^2)$. This approach is particularly inefficient if there are many regions but few intersections between regions. For this scenario, a significant amount of overhead will be introduced. Another problem is that it does not scale very well, except when there are many intersections between regions.

2.2.2 Grid-based Approaches

The grid-based approach [11, 18] is also known as the cell-based approach. The entire routing space is partitioned into a grid of cells. Each region is then mapped to the grid cells. A subscription region and an update region are assumed to overlap if they intersect with the same grid cell. The advantage of this approach is that the computational overhead is much less than that of the region-based approach and it is much more scalable. The main drawback of this scheme is that the overlap information is not accurate because entities will receive irrelevant information when their subscription regions cover the same grid cell as the update region, but the two regions do not overlap.

Obviously, some communication overhead will be introduced due to delivering irrelevant information; additional receiver-side filtering is also needed to filter the irrelevant data.

Figure 13 shows an example of the grid-based approach. In this example, U1 and S1 intersect with the same grid cell while U1 and S2 intersect with another grid cell. Therefore, the updates from the object associated with U1 will be routed to both S1 and S2. However, U1 and S2 do not actually overlap although they are assumed to overlap since they intersect with the same cell. S2 will receive irrelevant information and extra network resources will be consumed. Similarly, the updates from the object of U2 will be routed to S2 since U2 and S2 intersect with the same grid cell.

The grid cell size is an important factor in this approach. Large cell size may result in a large amount of irrelevant information transferred while for a smaller grid size, more resources will be consumed in maintaining the lists of each cell. Therefore, the choice of the cell size has a significant impact on the performance of the grid-based mechanism [24]. Some researchers have proposed a method to determine the optimal cell size [9].

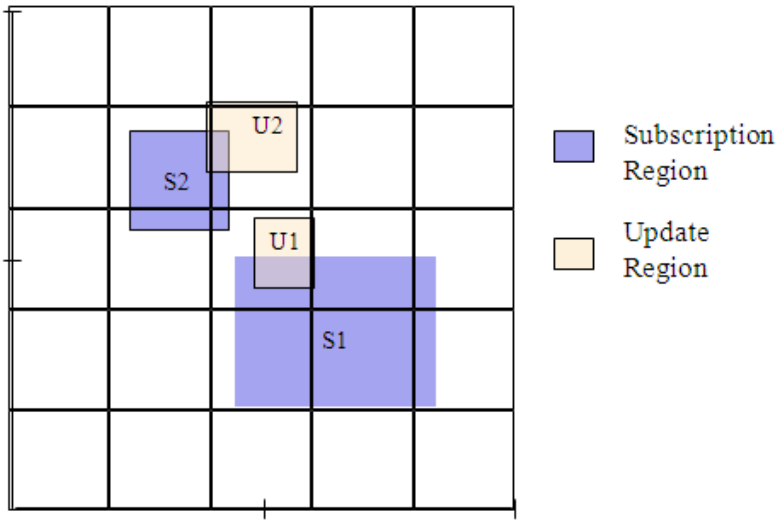


Figure 13. Grid-based Approach

2.2.3 Hybrid Approaches

Hybrid approach [10, 24] combines the region-based and grid-based approaches. It is an improvement over both approaches. It first maps all the regions to the grid cells with the grid-based approach. Then the region-based approach is used to determine an exact matching between update and subscription regions within each cell. For example in Figure 13, after using the grid-based algorithm, the updates from the object of U1 will be routed to both S1 and S2. However, U1 and S2 do not overlap although they intersect with the same cell. If the hybrid approach is used, the region-based approach should be used after the grid-based scheme is applied to compute the exact matching within each cell. Then the fact that U1 and S2 do not overlap will be examined and the updates from the object of U1 will only be routed to S1, which obviously overlaps with U1.

In this approach, the advantages of both the region-based and grid-based approaches are exploited. The overlap information is exact and the matching overhead is much lower than that of the region-based approach. The major problem is the same as that of the grid-based approach: the size of the grid cell is very important for the performance of the algorithm.

2.2.4 Sort-based Approaches

Sort-based algorithms [17, 21] are another popular approach for interest management. This algorithm has been used to compute the intersection between update and subscription regions. It projects the regions on each dimension and sorts the projections in order to determine the overlap information. After obtaining the overlap information on each dimension, the matching between update regions and subscription regions can be computed because two regions overlap if the projections of the two regions intersect along each dimension. For example in Figure 14, regions are projected

along x-axis and the projections are sorted for identifying the overlaps on x-axis between update and subscription regions.

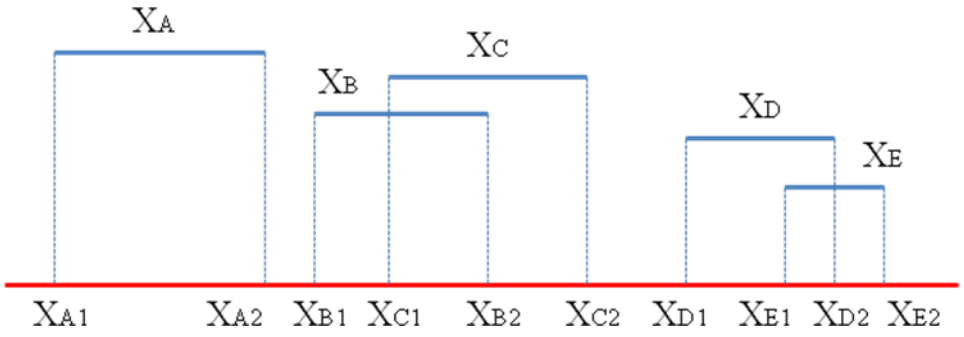


Figure 14.Projects of Regions to X-Axis

Raczy [21] proposed a sort-based interest management scheme. After projecting the regions on each axis, the scheme uses heap-sort to sort the projections. The computational complexity of such sorting is O(nlogn). An N-by-N bit-matrix is used to maintain the overlapping information assuming there are N update regions and N subscription regions. In Raczy's work, sort-based algorithm has been shown to yield favorable results compared with the region-based and hybrid algorithms. Pan [19] also proposed a sorted-based matching algorithm and argued that it has better storage and computational scalability than Raczy's algorithm in many cases.

2.3 Interest Management Scheme

In this section, we present our interest management mechanism for mobile peer-to-peer systems. We illustrate our sorting and matching algorithm first, and then present the mobile landmarking scheme for implementing the sorting and matching algorithm in mobile peer-to-peer systems.

2.3.1 Bucket Sort-based Algorithm

Our interest management scheme combines the advantages of grid-based and sort-based algorithms. This scheme first divides the multidimensional space into a grid of cells. Then a sort-based algorithm is used within each cell to compute the exact intersection between update and subscription regions. Specifically, our sort-based algorithm projects the regions on each dimension and uses bucket sort to sort the projections in order to find overlaps. This bucket sort-based algorithm is expected to have better computational efficiency than other algorithms based on comparison sorts if there are not many projection values falling into the same bucket. Our algorithm is applicable to dynamic sorting and matching of region modifications without resorting all regions and matching them again.

In the sort-based algorithm, a dimension reduction approach is used to reduce the multidimensional problem to a one-dimensional problem. A pair of regions overlaps if all of their projections on each dimension intersect. For example, in Figure 15, the projections of subscription region A and update region B on both the x-axis and y-axis have a non-empty intersection, so regions A and B overlap. The projections of subscription region C and update region D on the y-axis overlap while the projections of C and D on x-axis have no intersection, so regions C and D do not overlap.

The most important step in the scheme is to sort the projections on each dimension. For the regions residing in multiple cells, we need to record the lower and upper bounds of these regions for each cell. For example, the lower bound of subscription region E is $X_{E1(1)}$ and upper bound of E is $X_{E2(1)}$ in cell 1 for the projection sorting on x-axis, while the lower bound of E is $X_{E1(2)}$ and upper bound of E is $X_{E2(2)}$ in cell 2 ($X_{E2(1)}$ and $X_{E1(2)}$ are cell boundary values).

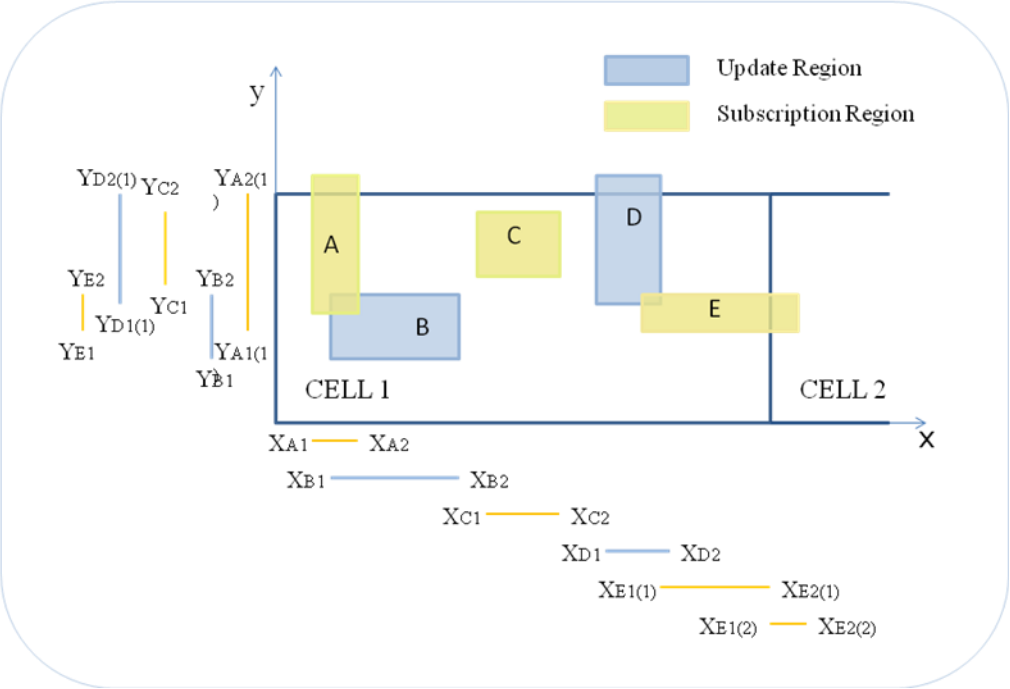
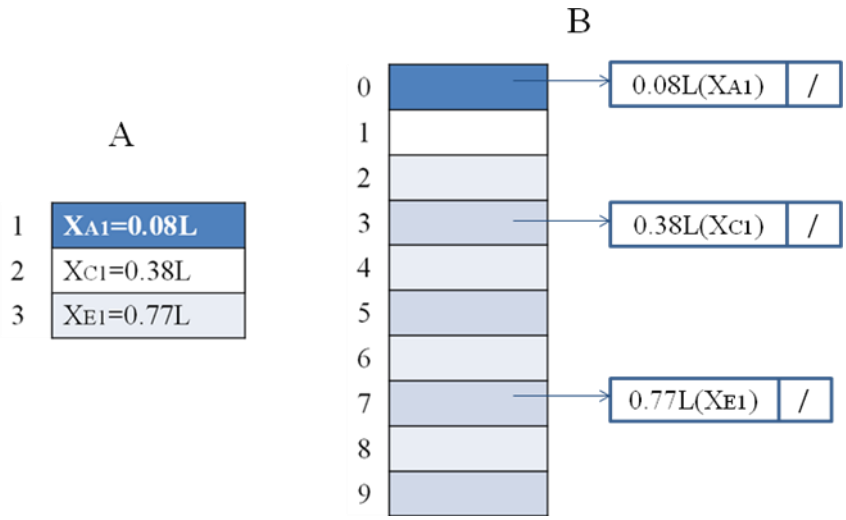


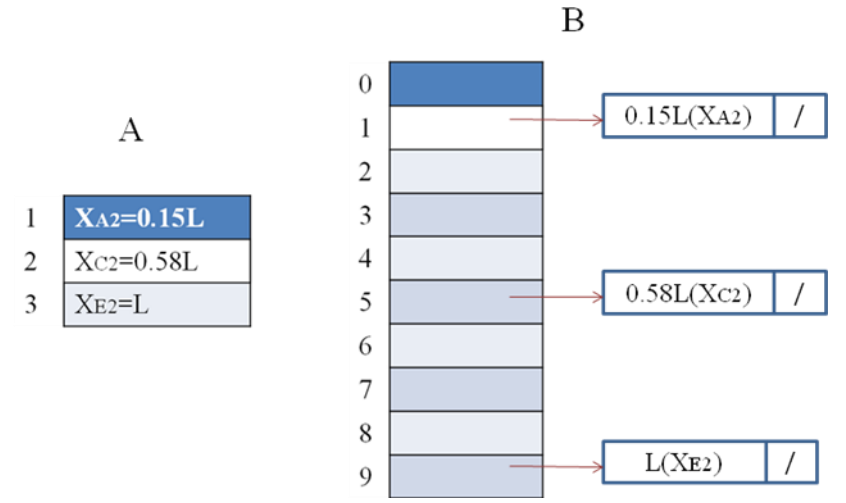
Figure 15. Dimension Reduction Approach

In our algorithm, we use bucket sort to sort the projections of subscription regions for each dimension. We store the lower bound and upper bound of all the subscription regions in one cell in an array, and then distribute these projections into the buckets. Bucket i holds values in the half-open interval [CellBoundValue+(i/k)*CellLength, CellBoundValue+((i+1)/k)*CellLength), where k is the number of buckets and CellBoundValue is the lower bound of each cell. For example, the CellBoundValue is 0 for cell 1, CellLength for cell 2, 2*CellLength for cell 3 and so on. Suppose CellLength is L and k is chosen to be 10, then bucket 0 for the cell 1 holds values in the interval [0, 0.1L). For the sorting of the elements in each bucket, since there are not many elements in each one, we can use bucket sort or other sort algorithms, such as insertion sort. The bucket sort-based algorithm works best if the distribution is uniform. In Figure 16(a), we show the bucket sort result for sorting the lower bounds of subscription regions A, C and

E in cell 1. The upper bounds can be inserted into buckets similarly and the result is shown in Figure 16(b).



(a)



(b)

Figure 16(a) and 16(b): Sorting the Projections of Subscription Regions

After distributing the projections of subscription regions into buckets, we need to process the update regions. If an update region does not overlap with a subscription region, there are only two possibilities:

(1) The lower and upper bounds of the subscription region are less than this update region, which means the upper bound of the subscription region must be less than the lower bound of the update region.
(2) The lower bound and the upper bound of the subscription region are greater than this update region, which means the lower bound of the subscription region must be greater than the upper bound of the update region.

Based on case (1), we first determine the bucket index if we insert an update region lower bound into the buckets used to sort subscription region upper bounds. Then all the subscription regions whose upper bounds are in the bucket preceding this bucket do not overlap with the update region; all the subscription regions whose upper bounds are in the same bucket but less than this update region lower bound do not overlap with this update region. Similarly, we can also determine the bucket index if we insert an update region upper bound into the buckets used to sort subscription region lower bounds. Then all the subscription regions whose lower bounds are in the bucket after this bucket do not overlap with the update region; all the subscription regions whose lower bounds are in the same bucket but more than this update region upper bound do not overlap with the update region.

Consider the above example illustrated in Figure 17. Suppose $X_{B1} = 0.12L$, $X_{B2} = 0.36L$, $X_{D1} = 0.62L$, $X_{D2} = 0.79L$. X_{C1} and X_{E1} (the lower bounds of subscription regions C and E) are more than X_{B2} (the upper bound of update region of B), so B does not overlap with C and E on the x-axis, as shown in Figure 17(a). Similarly in Figure 17(b), X_{A2} and X_{C2} (the upper bounds of subscription regions A and C) are less than X_{D1} (the lower bound of update region of D), so D does not overlap with A and C on the x-axis.

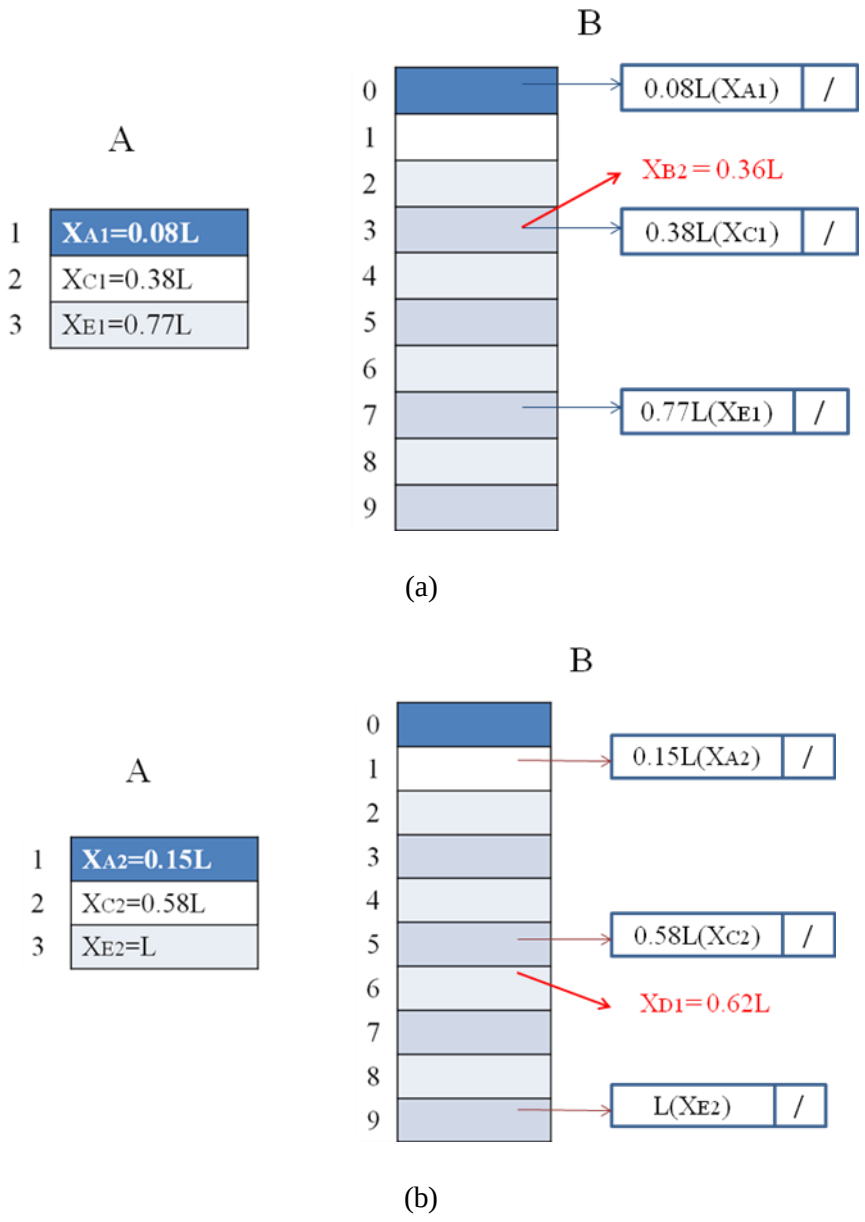


Figure 17(a) and 17(b): Determining Overlap Information

After we have determined if there are overlaps for each dimension, we can combine them to determine overlaps for the multidimensional space. The algorithm for static matching of all regions is shown in Figure 18. As mentioned before, some sort-based algorithms cannot deal with dynamic matching of a region modification without repeating the static matching of all regions. Here, if the modified region R is a subscription region, the original overlap information related to R and the projections of R

are removed. Then the scheme inserts the new bounds of R into the list of the corresponding sorting buckets. Then the new overlap information is obtained by comparing the lower bound of each update region with the upper bound of R as well as comparing the upper bound of each update region with the lower bound of R. If the modified region R is an update region, the original overlap information related to R is removed. Then the scheme determines the overlap information between R's new projections with the bounds of each subscription region. The algorithm for dynamic matching of one region modification is shown in Figure 19.

```
1.  for each subscription region
2.    for each dimension
3.    {
4.      insert its lower bound into SRegionLowerBoundSet;
5.      insert its upper bound into SRegionUpperBoundSet;
6.    }
7.  for each update region
8.    for each dimension
9.    {
10.     insert its lower bound into URegionLowerBoundSet;
11.     insert its upper bound into URegionUpperBoundSet;
12.   }
13.  for each dimension
14.  {
15.    insert each element in SRegionLowerBoundSet into the list of the corresponding bucket for
          lower bound sorting;
16.    insert each element in SRegionUpperBoundSet into the list of the corresponding bucket for
          upper bound sorting;
17.    for each bucket
18.          sort the list with insertion sort;
19.   }
20.   for each dimension
21.  {
22.     for each update region lower bound in URegionLowerBoundSet
23.     {
24.       determine the bucket index if this update region lower bound is inserted into the
                buckets used to sort subscription region upper bounds;
25.       all the subscription regions whose upper bounds are in the bucket before this bucket
                do not overlap with this update region;
26.       all the subscription regions whose upper bounds are in the same bucket but less than
                this update region lower bound do not overlap with this update region;
27.    }
28.  }
29.   for each dimension
30.  {
31.    for each update region upper bound in URegionUpperBoundSet
32.    {
33.      determine the bucket index if this update region upper bound is inserted into the buckets
              used to sort subscription region lower bounds;
34.      all the subscription regions whose lower bounds are in the bucket after this bucket do not
              overlap with this update region;
35.      all the subscription regions whose lower bounds are in the same bucket but more than
              this update region upper bound do not overlap with this update region;
36.    }
37.  }
```

Figure 18. Algorithm for Static Matching of All Regions

```
1.  if (R is a subscription region)
2.  {
3.    for each dimension
4.    {
5.      remove the overlap information related to subscription region R;
6.      remove the original lower bound and original upper bound of R in the list of the
            corresponding sorting bucket;
7.      insert the new lower bound and new upper bound of R in the list of the corresponding
            sorting bucket;
8.    }
9.    for each dimension
10.     for each update region
11.     {
12.       if the lower bound of the update region is more than the upper bound of R,
13.         this update region does not overlap with R;
14.       if the upper bound of the update region is less than the lower bound of R,
15.         this update region does not overlap with R;
16.     }
17. }
18. else                    // if R is an update region
19. {
20.     for each dimension
21.       remove the overlap information related to update region R;
22.   for each dimension
23.   {
24.     determine the bucket index if the lower bound of R is inserted into the buckets used
            to sort subscription region upper bounds;
25.     all the subscription regions whose upper bounds are in the bucket before this bucket
            do not overlap with R;
26.     all the subscription regions whose upper bounds are in the same bucket but less
            than this update region lower bound do not overlap with R;
27.   }
28.   for each dimension
29.   {
30.     determine the bucket index if the upper bound of R is inserted into the buckets used
            to sort subscription region lower bounds;
31.     all the subscription regions whose lower bounds are in the bucket after this bucket
            do not overlap with R;
32.     all the subscription regions whose lower bounds are in the same bucket but more
            than this update region upper bound do not overlap with R;
33.   }
34. }
```

Figure 19. Algorithm for Dynamic Matching of One Region Modification

2.3.2 Mobile Landmarking

When dividing the entire space into cells and sorting within each cell in a mobile peer-to-peer system, one important question concerns where the sorting and matching computation is performed. Conventional landmarking technology [22] suffers from the limitation that it assumes a set of fixed, stationary landmark nodes. All entities are expected to know the landmark nodes and communicate with other entities through the landmark nodes. This requires a server, but in mobile peer-to-peer systems there are usually no sets of fixed nodes available. Therefore, we introduce the mobile landmarking concept into mobile peer-to-peer systems.

This scheme works without any fixed landmark nodes. Instead, it uses a set of landmark keys. For example, a landmark key can be defined as the center point of a cell. Rather than having dedicated landmark nodes, those nodes which are currently closest to one of the landmark keys become mobile landmark nodes. When one of the current mobile landmark nodes leaves this cell, another node (which is now closest to the landmark key) will automatically become the new mobile landmark node, and the original mobile landmark node should transmit the overlapping information in that cell to the new mobile landmark node.

An example is shown in Figure 20. The entire environment is partitioned into sixteen cells and each landmark key is responsible for each cell. The node which is currently closest to one of the landmark keys becomes a mobile landmark node. In cell 16, the current mobile landmark node resigns and another node which is now closest to the landmark key becomes the new mobile landmark node automatically.

At the initialization stage, each entity will send a message about their subscription and update regions to the mobile landmark node responsible for the cell in which the entity resides. The data format of the message is (EntityID, EntityLocation,

URLowerBound, URUpperBound, SRLowerBound, SRUpperBound), where URLowerBound refers to the lower bound of the update region, URUpperBound

Figure 20. Mobile Landmarking

refers to the upper bound of the update region, SRLowerBound refers to the lower bound of the subscription region, and SRUpperBound refers to the upper bound of the subscription region. Then the mobile landmark node performs the static sorting and matching. The mobile landmark node then sends a message to each entity in the cell about other entities whose subscription regions overlap with this entity's update region. The data format of the message is (EntityID1, EntityLocation1, EntityID2, EntityLocation2, EntityID3, EntityLocation3,…). When entities receive the information, they will transmit state update messages to these entities listed in the message. In this scheme, we assume all communications are reliable.

When there is an update or subscription region modification, the entity will send the modification information to the mobile landmark node responsible for the cell in

which the entity resides. However, it will incur too much communication overhead if an entity sends the updates each frame. We can have each entity send the modification information periodically or in a space-driven manner, where an entity sends the region modification information to the mobile landmark node only when its position shifts from the last update by an amount exceeding a DistanceThreshold in either dimension. The data format of the message is (EntityID, EntityLocation, OldURLowerBound, OldURUpperBound, OldSRLowerBound, OldSRUpperBound, NewURLowerBound, NewURUpperBound, NewSRLowerBound, NewSRUpperBound).

After the mobile landmark node receives the message, it will perform the dynamic matching of this region modification. If the modified region is an update region, the mobile landmark node will return a message to the entity about the subscription regions that overlap with either the original update region or the new update region but not both. The data format is (EntityID1, EntityLocation1, Removed, EntityID2, EntityLocation2, Removed,…, EntityIDk, EntityLocationk, Added, …). The removed entity refers to the entity whose subscription region overlaps with the original update region while the added entity refers to the entity whose subscription region intersects with the new update region. Then the entity will transmit its update information to the overlapped subscription regions. If the modified region is a subscription region, the mobile landmark node will record the information of update regions that overlap with either the original subscription region or the new subscription region but not both and send the information to the entities to which these update regions belong. The data format of the message to each update region is (EntityID, EntityLocation, Removed) if the update region only overlaps with the original subscription region or (EntityID, EntityLocation, Added) if the update region only intersects with the new subscription region. Then the affected entities will send the update information based on the new overlap information. The algorithm for mobile landmark node is shown in Figure 21.

Since peer nodes communicate continuously with mobile landmark nodes, they will have knowledge of the mobile landmark nodes. For that purpose, each mobile landmark node sends beacons periodically within the cell in which this mobile landmark node resides. Whenever a node hears a beacon, it stores the information of this mobile landmark node. Nodes periodically examine whether they have moved closer to a new mobile landmark node, for example, when they have moved into a new cell.

```
At Initialization:
1.  Receive information from peers in the cell about the subscription and update
      Regions;
2.  Sort all the regions and perform the static matching;
3.  Send a message to each peer in the cell about the overlap information;

After One Region Modification:
4.  Receive region modification information from the peer in the cell;
5.  if (the modified region is an update region)
6.  {
7.     Perform the dynamic sorting and matching for update region modification;
8.     Send back a message to the peer about the subscription regions who
           overlap with either the original update region or the new update region
           but not both;
9.  }
10. else            // the modified region is a subscription region
11. {
12.   Perform the dynamic sorting and matching for subscription region
           modification;
13.   Send a message to each peer whose update region overlap with
           either the original subscription region or the new subscription
           region but not both;
14.  }
```

Figure 21. The Algorithm for Mobile Landmark Nodes

2.4 Experiments and Results

To evaluate the performance of the proposed interest management mechanism, a mobile peer-to-peer transportation system is simulated as a routing agent in the ns2 simulator. All simulations that we carried out modeled wireless networks over the course of one (simulated) hour. The simulated system is a two-dimensional space of 1000

distance units by 1000 distance units. Its input is the traces obtained from VISSIM (PTV Vision VISSIM), a microscopic multi-modal transportation simulator. The simulated traffic network has 10*10 intersections with each intersection under two-phase signal control. The cycle length of the traffic signal is 120 seconds. Vehicles entering an intersection will maintain a straight path with 95% probability, or turn left with 2% probability or turn right with 3% probability. Each roadway is a two-way arterial, with one lane in each direction. Vehicles enter this network from each entrance (in total 40 entrances) at initialization.

In this simulated mobile peer-to-peer transportation system, vehicles communicate with each other without infrastructure. Each vehicle is associated with a update region and a subscription region. Our scheme is used to obtain the overlap information between update regions and subscription regions in order that each vehicle only receives the information of interest. We conducted three sets of experiments to evaluate our scheme: Comparison of communication cost with other schemes; Comparison of execution time of static matching using different schemes; Comparison of execution time of dynamic matching under region modifications with other schemes. For the first set of experiments, the range per dimension is randomly generated with a uniform probability density function between 50 distance units and 150 distance units. We use the space-driven manner when sending the region modification information and the DistanceThreshold is chosen to be 10 distance units here. For the second and third set of experiments, the region definition is related to overlapping rate, which will be defined below. All our experiments run on a single PC with Intel (R) Core(TM) 2 Duo CPU P7550 2.26GHz, 2GB RAM and Windows XP OS.

2.4.1 Comparison of Communication Cost

First, we evaluate the communication cost of our scheme and compare it with another three schemes: region-based, cell-based and hybrid schemes. The communication cost refers to the number of messages that are sent by all vehicles at each frame. The grid cell size for the cell-based scheme is 100 distance units in this set of experiments.

As is shown in Figure 22, the communication cost increases linearly with the number of vehicles, as expected. The cell-based scheme has the highest communication cost. The reason is that the overlapping information computed by the cell-based algorithm is not exact, and this lack of precision creates unnecessary data communication. A relatively large grid cell size also causes more irrelevant data to be received by entities. The hybrid scheme has lower communication cost than the cell-based and region-based schemes. That is because the hybrid scheme reduces the communication cost in the region-based mechanism by utilizing the cell-based mechanism. The figure also shows that the communication cost of the proposed scheme is slightly higher than that of the hybrid approach, which suggests that the mobile landmarking design does not introduce an excessive amount of communication overhead in order to improve the computation performance. Although the mobile landmark node periodically sends beacons, the beacon message is restricted to be transmitted within the cell rather than the entire system. Therefore, the communication cost is only slightly higher than that of the hybrid approach.

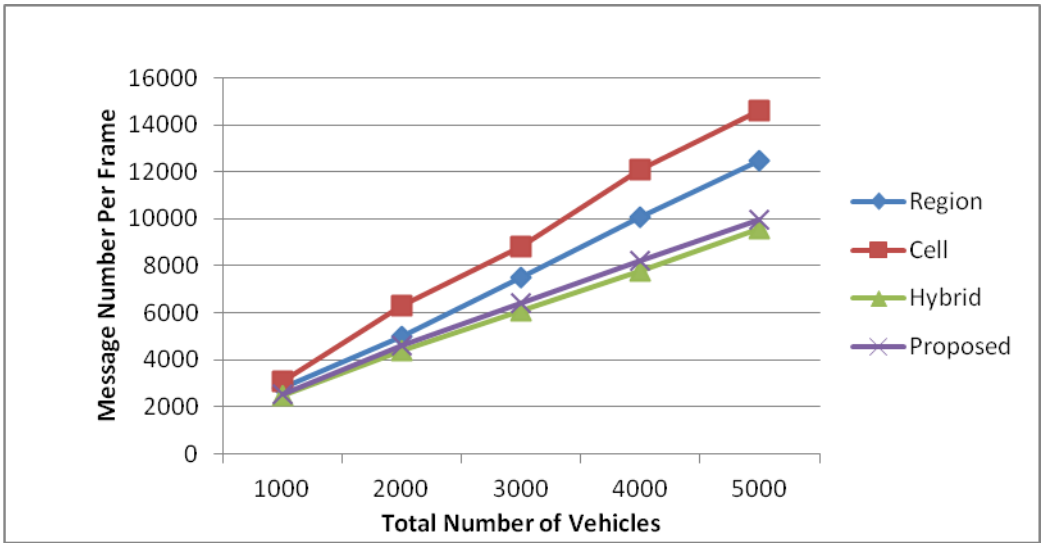


Figure 22. Comparison of Message Number

Since our proposed approach and hybrid approach have similar performance, we would like to use a paired t-test to access statistically the difference between the two means. The test statistic is calculated as:

$$D_i = Y_{1i} - Y_{2i} (i = 1, 2, ..., n)$$

$$\overline{D}_n = \frac{1}{n} \sum_{i=1}^{n} D_i$$

$$S_n^2(D) = \frac{1}{n-1} \sum_{i=1}^{n} (D_i - \overline{D}_n)^2$$

$$t = \frac{\overline{D}_n}{\sqrt{\frac{S_n^2(D)}{n}}}$$

The sample size is 100 and the degree of freedom is 99. Table 1 displays the results.

Table 1. Paired t-test for Two Means of Message Numbers

Vehicle Number	1000	2000	3000	4000	5000
t statistic	1.5679	1.6991	1.8326	1.9958	2.0319
p-value	0.1201	0.0924	0.0699	0.0487	0.0448

From the results, the p-value associated with t is high (> 0.05) when the total vehicle number is less than 4000, which indicates that there is no significant difference between the two means. When the vehicle number is increasing beyond 4000, the p-value decreases to about 0.05 indicating that the means may be different. This makes sense because when the traffic network becomes more congested, more communication overhead will be introduced due to mobile landmarking design.

2.4.2 Comparison of Execution Time of Static Matching

In this set of experiments, each scheme (region, hybrid, sort and our proposed scheme) performs static matching of all regions and the execution time of static matching is recorded. One of the important factors affecting the performance is the overlapping rate, which is defined as follows.

$$Overlapping\ rate = \frac{\sum area\ of\ regions}{area\ of\ the\ entire\ space}$$

If the space is 1000*1000, and one region is 10*10, where the number of regions is fixed at 1000, the overlapping rate is

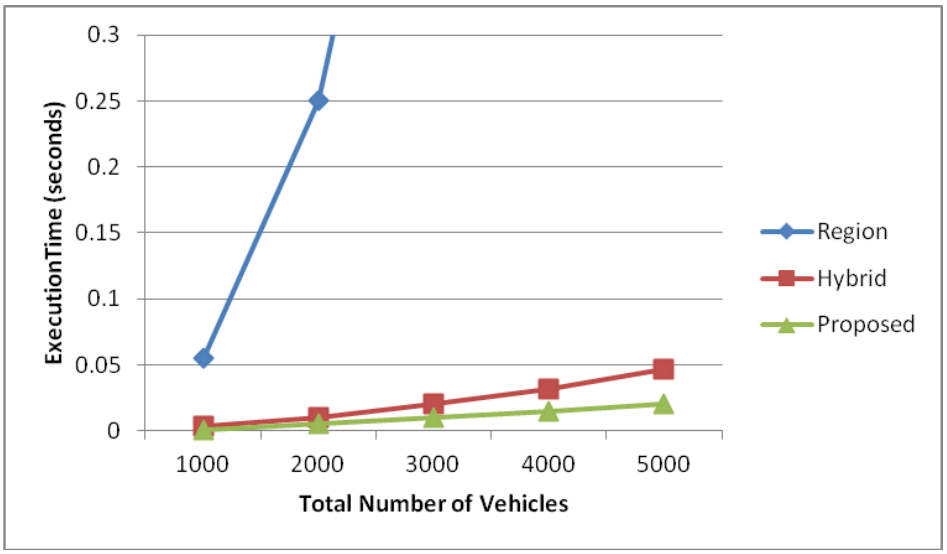
$$0.1 = \frac{1000 * (10 * 10)}{1000 * 1000}$$

The higher overlapping rate implies greater probability of region overlap. Two different values 0.01 and 1 are used in the experiments.

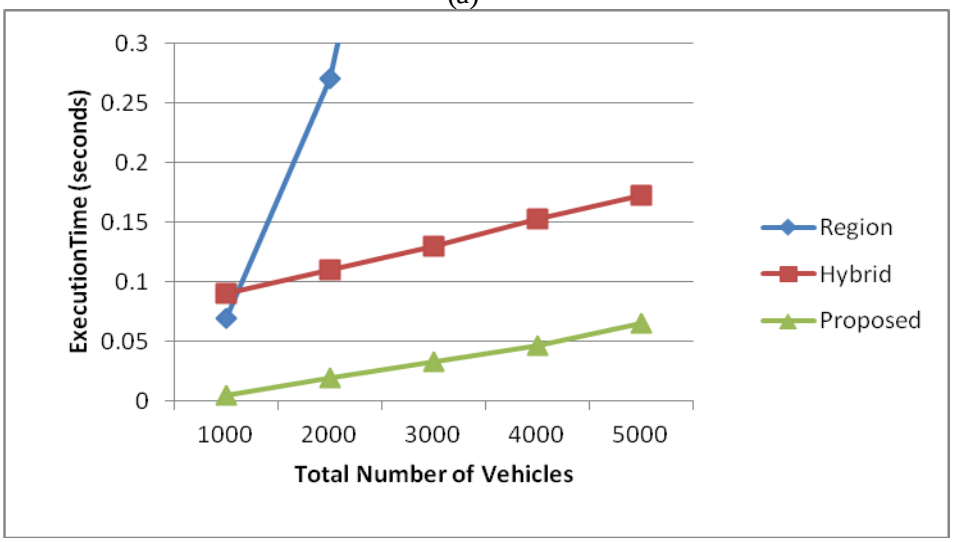
Figure 23(a) shows the execution time of three schemes with the overlapping rate set to be 0.01. The hybrid and the proposed algorithms have similar performance, while the region-based approach yields very poor results, especially when the number of vehicles is increased. Both the proposed algorithm and the hybrid algorithm with grid cell size 100 distance units perform well because the regions are very small and usually they do not cover more than one cell when the overlapping rate is 0.01. The execution time of

the proposed algorithm is half of that of the hybrid approach. Hence the proposed scheme is the best choice among the three schemes in this scenario.

Figure 23(b) shows the experiment results when the overlapping rate is 1. Our proposed algorithm also performs best with higher overlapping rate because it is less affected by the overlapping rate. The hybrid algorithm with grid cell size 100 distance units has high overhead when the regions become large and covering more than one cell. The region-based algorithm still shows poor results because it exhaustively compares all regions to determine the overlap information.



(a)



(b)

Figure 23. Comparison of Execution Time of Static Matching

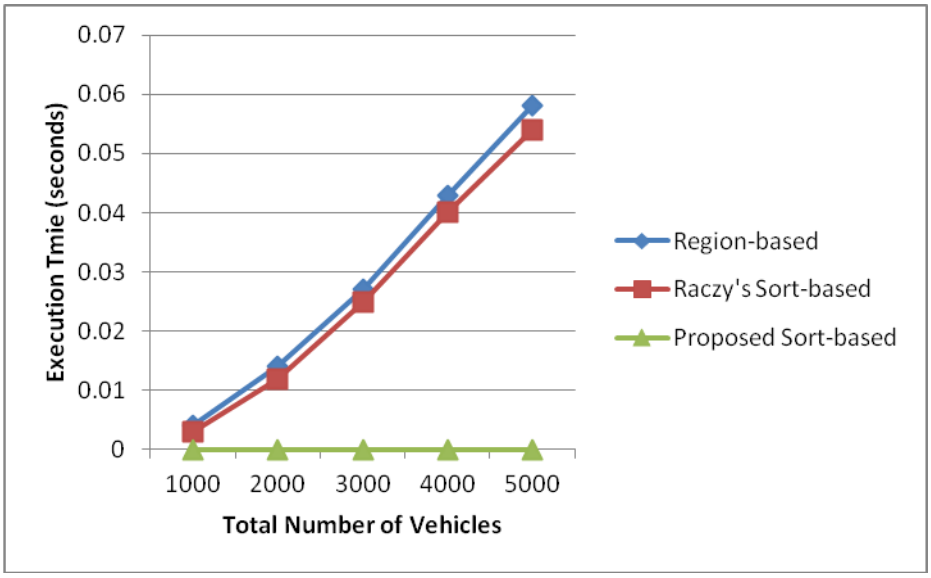
Since our proposed approach and hybrid approach have similar performance when the overlapping rate is 0.01, we also used a paired t-test to examine whether there is significant difference between the two means. Table 2 shows the results for an increasing number of vehicles. From the results, the p-values are relatively low (< 0.05) for all the cases, which indicates significant difference in means across the two designs. Our proposed approach is more efficient than hybrid approach.

Table 2. Paired t-test for Two Means of Execution Time

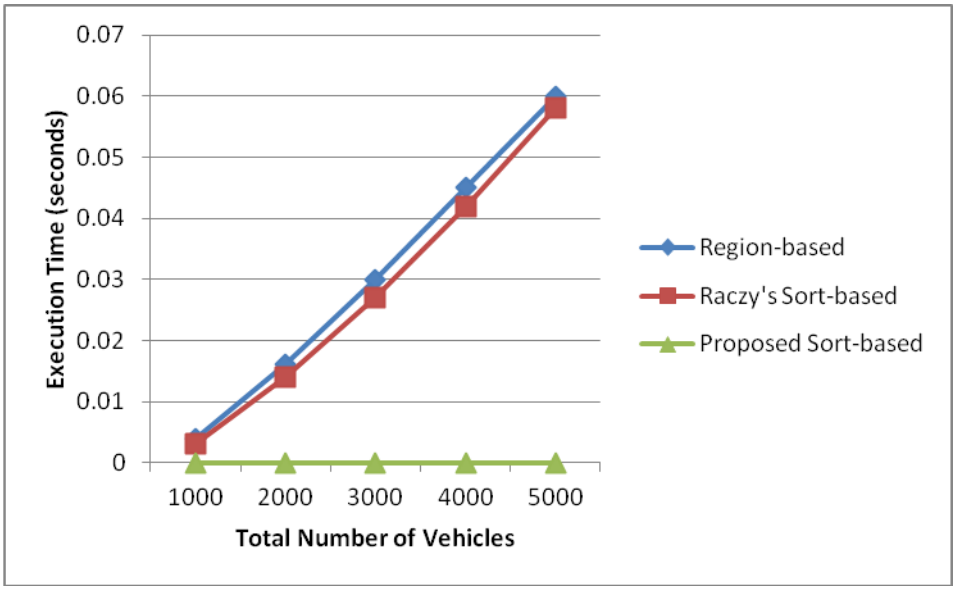
Vehicle Number	1000	2000	3000	4000	5000
t statistic	1.9987	2.1567	2.4851	2.8803	3.0613
p-value	0.0484	0.0335	0.0146	0.0049	0.0028

2.4.3 Comparison of Execution Time of Dynamic Matching

For dynamic matching, a random subscription or update region is modified in one iteration and each scheme (region-based algorithm, Raczy's sort-based algorithm [21] and the proposed algorithm) performs dynamic matching of a region modification for 100 iterations and calculates the average performance of the 100 iterations. Figure 24 shows that the dynamic matching time of region-based and Raczy's sort-based algorithm is almost the same as the static matching time of all regions. This is because they cannot perform dynamic matching without processing all the regions again. The dynamic matching time of the proposed algorithm is almost zero no matter whether the overlapping degree is 0.01 or 1 because our proposed algorithm only removes the overlap information related to original region bounds and obtains the overlap information related to new region bounds without resorting all the regions.



(a)



(b)

Figure 24. Comparison of Execution Time of Dynamic Matching

2.5 Non-Uniform Distribution for Bucket Sort

For the non-uniform distribution, if we use bucket sort as usual, the performance may decrease dramatically. Therefore, we use a three-phase process to sort all the projections of subscription regions and update regions. The first phase is called random sampling, which is used to determine the boundaries of the bucket sort. The second phase

uses the bucket boundaries obtained in the first phase to sort the regions. We hope to get equi-depth buckets also after putting all the data items into buckets. The third phase keeps splitting and merging buckets during runtime to maintain equi-depth buckets.

The approach to determine the boundaries of buckets can be derived from algorithms for equi-depth histograms. Given a data set, the B-bucket equi-depth histogram algorithm seeks to find a sorted sequence of B − 1 boundaries over the sorted list of the data, such that the number of data between each two consecutive boundaries is approximately N/B, where N is the data set size [75]. This goal can only be achieved within a certain approximation given time and space constraints.

One well-known approach is to use random sampling [73]. The idea is to obtain n (n < <N) samples from the data set randomly and uniformly, without replacement. (Such a sample is called a simple random sample.) Samplesort [74] is a popular approach using random sampling. It uses the partitioning paradigm: it uses a set of *m* pivots sampled from the given set to partition *n* keys. It selects and sorts m < n pivots, and then associates a bucket with each interval dictated by the pivots. The next step is to use binary search to distribute the rest of the keys into buckets (m + 1 buckets). We use similar procedures in our algorithm as samplesort. The first step is to use an efficient algorithm [76] to do the random sampling. The population size N can be unknown at the beginning of the algorithm. At the conclusion of the algorithm, N is contained in k as follows and n random samples are obtained, provided that n≤N [76].

1. Set k <- 0.
2. Are there any remaining members in the population? If no, terminate. Otherwise, obtain the next member and set k <- k+1.
3. (a) If k<n or k=n, the *k*th population member becomes the *k*th member of the sample. Go to Step 2.

(b) If k>n, generate a uniform (0,1) random variable U and set j <- 1+ [U*k*] where [.] denotes the integer part. If j<n or j=n, the *j*th member of the current sample is replaced by the *k*th population member. Go to Step 2.

Now we need to compute the sample size which is sufficient enough to make the sample distribution close to the underlying distribution.

If the sample distribution is close to the underlying distribution, then

$$\frac{|S_t|}{|S|} = \frac{|X_t|}{|X|} \pm \varepsilon$$

where $|S_t|$ denotes the number of the data items in the bucket *t* after random sampling, $|X_t|$ denotes the number of the data items in the bucket *t* for the whole data set, $|S|$ denotes the total number of samples and $|X|$ denotes the number of all the data items.

For all the data items, we hope we can get equi-depth buckets after putting them all into the buckets to make sure the running time is linear, which means $\frac{|X_t|}{|X|} = \frac{1}{k}$ (*k* is the bucket number). We know that $\frac{|S_t|}{|S|} = \frac{|X_t|}{|X|} \pm \varepsilon$ and we can define $\varepsilon = \frac{1}{10k}$ in our experiment.

We define the following indicator variable:

$$I_i^t = \begin{cases} 0 & 1-p \\ 1 & p \end{cases}$$

which means the probability of *i*th item in the sample falling into the *t*th bucket is *p*, then

$$\frac{|S_t|}{|S|} = \frac{\sum_{i=1}^{m} I_i^t}{m}$$

$$E(\frac{|S_t|}{|S|}) = p$$

where m= $|S|$ is the number of the data items in the sample.

Based on the Hoeffding Bounds (Concentration Inequalities), we have

$$P_r[\left|\frac{|S_t|}{|S|}-\frac{|X_t|}{|X|}\right|>\varepsilon]=P_r[\left|\frac{\sum_{i=1}^{m} I_i^t}{m}-\frac{1}{k}\right|>\varepsilon]\le e^{-2m\varepsilon^2}$$

If we want $P_r[\left|\frac{\sum_{i=1}^{m} I_i^t}{m}-\frac{1}{k}\right|>\varepsilon]\le \alpha=1-\delta$, then $e^{-2m\varepsilon^2}\le 1-\delta$

After computation, we get the sample size $m\ge\frac{\ln(1-\delta)}{-2\varepsilon^2}$, which means we have $(\delta\cdot 100\%)$ confidence to say that the sample size $m\ge\frac{\ln(1-\delta)}{-2\varepsilon^2}$ is sufficient enough to make the sample distribution close to the underlying distribution.

The third phase is regarding the runtime updates of bucket boundaries. The approach is based on merging and splitting buckets [77]. We record the number of data items in each bucket and when there are updates about subscription regions or update regions, we revise the count for affected bucket also. When a bucket count reaches the threshold T (T = (1+β)N/k, where N is the size of the whole data set, K is the bucket number and β>0 is a tunable performance parameter), we split the bucket in half. We can also merge some two adjacent buckets whose total count does not exceed the threshold T [77].

We conduct experiments to evaluate the effectiveness and efficiency of our proposed approach and examine the impact of different β value on performance. We compare the execution time of original approach and current approach with random sampling and runtime merging and splitting. Figure 25 shows the execution time of 500 region modifications under different vehicle numbers. Current approach takes less

execution time than original approach no matter whether the β value is low or high. This indicates that current approach with random sampling and runtime merging and splitting increases the algorithm efficiency. When β value is 0.6, the execution time is the lowest among three scenarios. We explore the impact of β value on execution time and operation number further in the following experiments.

Figure 26 shows the impact of different β values on execution time. When the β value is very low, such as 0.2, the execution time is relatively high since many merging and splitting operations will be needed when the threshold T is low, which will introduce operation overhead. When the β value increases from 0.6 to 1.4, execution time increases also, since equi-depth buckets are difficult to maintain if the threshold T is high. The buckets will become unbalanced if insufficient merging and splitting operations are used. Figure 27 shows the number of merging and splitting operations with different β values. The operation number decreases sharply when β value increases from 0.2 to 0.6. After the β value reaches 1.0, operation number almost remains the same. Based on the experimental results in Figure 26 and Figure 27, 0.6-0.8 is a good tradeoff for β value.

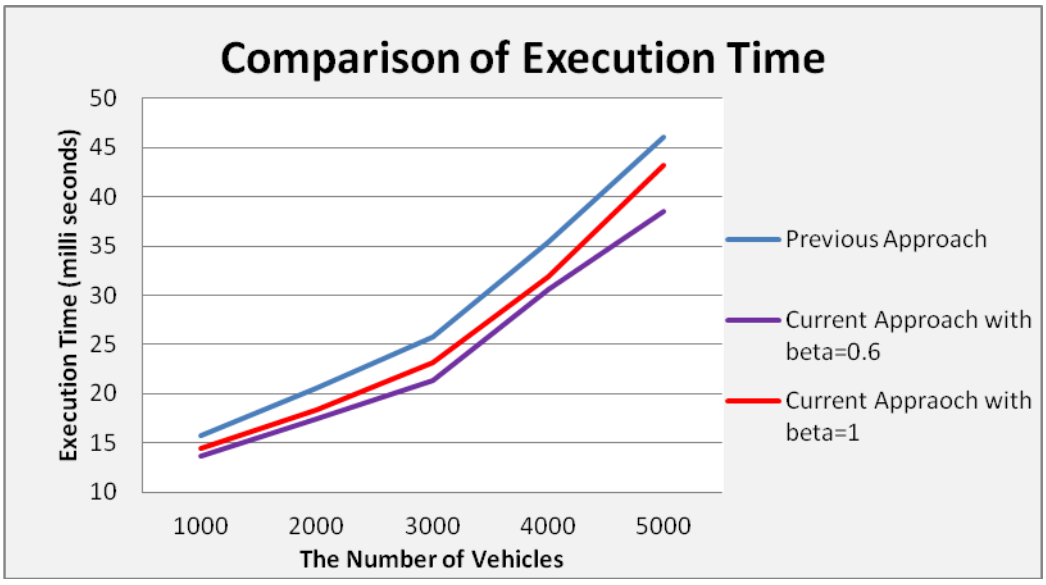


Figure 25. Comparison of Execution Time of Original and Current Approaches

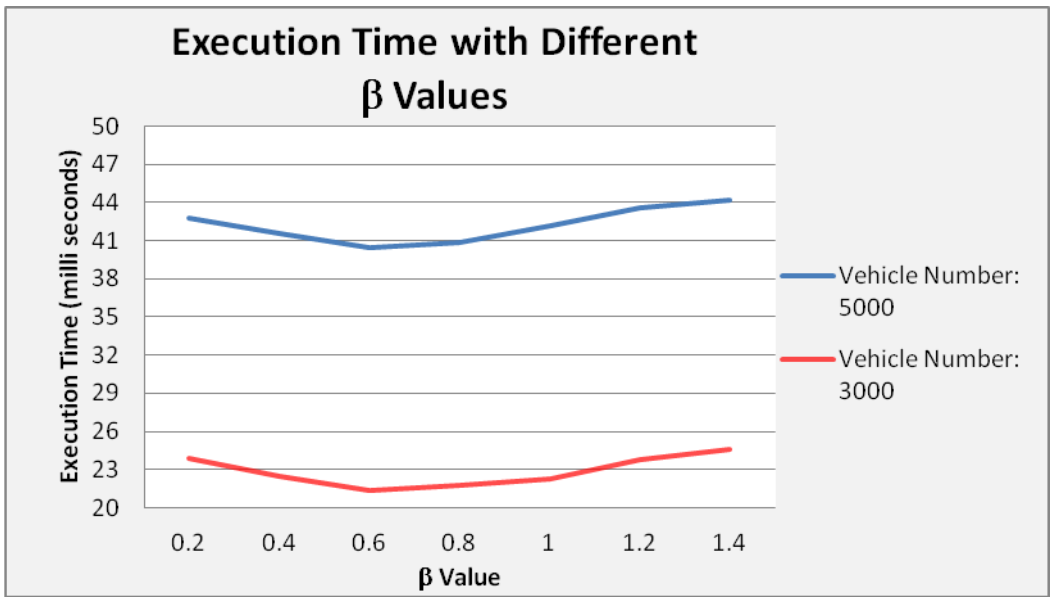


Figure 26. Comparison of Execution Time with Different β values

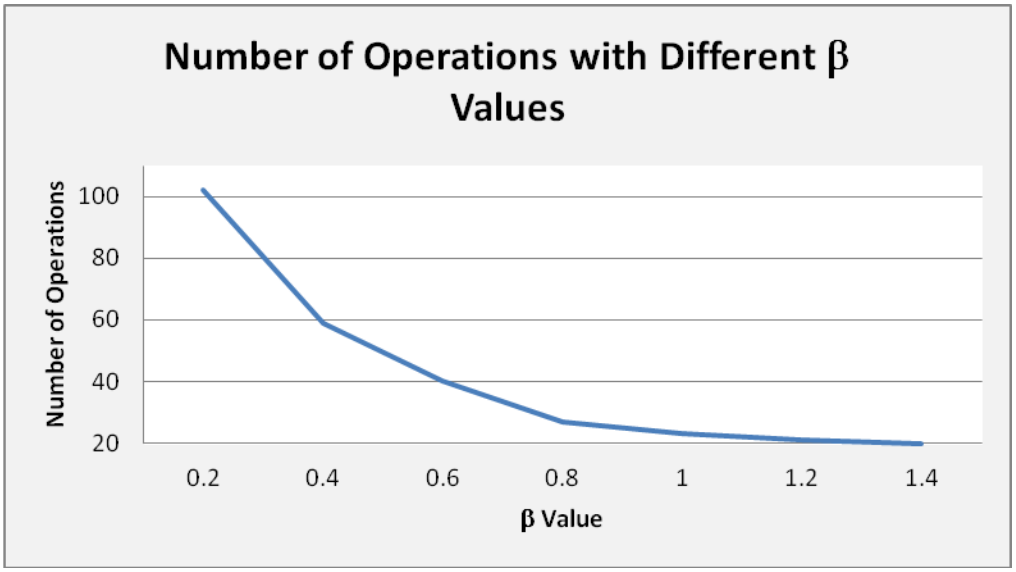


Figure 27. Comparison of Operation Number with Different β values

2.6 Conclusion

In this work, we have proposed an interest management scheme for mobile peer-to-peer systems, that divides the entire space into cells and uses bucket sort to sort the regions in each cell. The interest management scheme is applicable to dynamic sorting and matching of region modifications. If region modifications occur, the algorithm does not need to re-sort the projections and conduct the entire matching again. The bucket sort-based scheme is expected to have better computational efficiency than other algorithms based on comparison sorts. We also introduce the mobile landmarking design to implement this sort-based scheme in mobile peer-to-peer system simulation. The design does not have servers, but every peer can become a mobile landmark node to take the server-like role to sort and match the regions. To evaluate the new scheme, a mobile peer-to-peer 10*10 transportation system is simulated. The performance results show that the scheme has better computational efficiency for both static matching and dynamic matching. Furthermore, it is also shown that the mobile landmarking design does not introduce a lot of communication overhead in order to improve the computation performance.

CHAPTER 3

TRAVEL TIME PREDICTION MODEL

3.1 Introduction

Travel time prediction is one of the most important problems in intelligent transportation system research. It allows the travelling public to be proactive in routing decisions and planning. Researchers have proposed various types of prediction models. Regression-based approaches [40] make use of variables closely related to travel time, such as traffic flow, speed and occupancy to predict travel time. Other traditional models include time series [38] and Kalman filtering models [39]. These techniques explore the characteristics of historical data and utilize them to predict future values. Recently, many machine learning techniques have been examined. For instance, Artificial Neural Network (ANN) [37, 43] is a popular method, and many variations of basic models have been studied. Other techniques include support vector machine (SVM) [44], Bayesian [42] and K-Nearest Neighbors (KNN) [41].

Although several traditional approaches examine the characteristics of traffic flow, achieving high prediction accuracy has been elusive. Bayesian and KNN are basic machine learning approaches that must usually be combined with other methods such as ANN to obtain accurate prediction results. For SVM, it is important but very difficult to select proper kernel functions and optimal parameters. Theoretically, ANN can capture any relationship between output and input values, however, it can suffer from over-fitting, and prediction accuracy can also be improved.

Boosting is a very effective machine learning technology for classification [45, 48, 51, 52] and regression [46, 47, 52]. To our knowledge, it has not been previously applied to the travel time prediction problem. The proposed algorithm considers travel time

prediction as a classical regression problem, and uses a neural network as the basic learner to capture the characteristics of traffic. We then use boosting to augment this learning approach to increase prediction accuracy.

Collecting raw traffic data such as vehicle speed is very important for any prediction approach. When data are transmitted from sensors or probe vehicles to other vehicles, bandwidth resources will be consumed. It is reasonable to expect that increased collection frequency results in more accurate predictions, but bandwidth is a limited and expensive resource, especially in wireless networks. Therefore, finding a suitable minimum collection frequency that achieves good accuracy is important. In this thesis, we also explore the relationship between data collection frequency and travel time prediction accuracy. We use a binary search-like approach to compute the lower bound on the collection frequency while maintaining optimal prediction accuracy. To the authors' knowledge, this is the first attempt to consider QoS factors (bandwidth) in travel time prediction.

The contributions of this work are threefold. First, the boosting approach is introduced as a method for travel time prediction in conjunction with neural network models to help capture characteristics of traffic and increase prediction accuracy. Second, we examine the relationship between data collection frequency and travel time prediction accuracy, and propose an approach to compute the lower bound of the collection frequency while maintaining high prediction accuracy, thereby introducing QoS as a factor in travel time prediction for the first time. Finally, we explore the influence of various factors on prediction accuracy such as monitoring interval time for historical data and the number of iterations used by the boosting algorithm.

The rest of this chapter is organized as follows: related work about travel prediction models is introduced first. The boosting-based travel time prediction approach is then described in detail. Experimental results are presented in Section 3.4. Discussions

about the basic learner for boosting are presented in Section 3.5. Finally, the chapter is concluded in Section 3.6.

3.2 Related Work

Several researchers have proposed various types of travel time prediction models. Among these models, time series models and machine learning models are two popular approaches. Most studies show that prediction accuracy is often compromised by the underlying mechanism of prediction models more than other influencing factors [53]. Several prediction methodologies and techniques are presented as follows.

3.2.1 Time Series Models

Travel time can be predicted from historical data by analyzing traffic information from fields. For instance, the traffic data, such as the speed, direction and traffic volume are one example of historical data. Various techniques, such as statistical approaches and mathematical methods can then be used for developing the travel-time prediction model. One method used by many researchers [78, 79, 80] is stated as follows:

$$T^*(t,\Delta) = \sum_{l=0}^{L-1} \frac{x_{l+1} - x_l}{v(x_l, t-\Delta)} \tag{1}$$

where $x_{l+1} - x_l$ is the link length, $v(x_l, t-\Delta)$ is the speed at the start of the link and L is the total number of links. This approach is intuitively appealing since $T^*(t, \Delta)$ is based on the available data that are closest to t temporally. $T^*(t, \Delta)$ represents our initial guess of $T(t)$ given the available data $v(x_l, t-\Delta)$.

Based on this method, we use the speed information at the same time point for each link. For example, if we want to predict the speed $v(t)$, we will use the speed information at the following time series: t-nk, t-$(n-1)$k, …, t-2k, t-k for each link. For link l, $v(x_l, t)$ is computed based on v(x_l, t-nk), v(x_l, t-$(n-1)$k), …, v(x_l, t-2k), v(x_l, t-k). For

link $l+1$, $v(x_{l+1}, t)$ is computed based on $v(x_{l+1}, t-nk)$, $v(x_{l+1}, t-(n-1)k)$, …, $v(x_{l+1}, t-2k)$, $v(x_{l+1}, t-k)$.

In the above equation, $v(x_l, t-\Delta)$ is the predicted speed based on historical data, such as $v(x_l, t-\Delta-nT_0)$, $v(x_l, t-\Delta-(n-1)T_0), \ldots, v(x_l, t-\Delta-2T_0)$, $v(x_l, t-\Delta-T_0)$ (T_0 is the update period). Now the question is how to predict the speed.

Let T_0 be the period of speed updating, namely a speed data point is generated every T_0 time units, and f_0 is the data update frequency ($f_0 = 1/T_0$). We want to predict v_{n+1} based on historical speed data measured by sensors, and using v_{n+1} to predict the traffic time. Suppose we set the monitoring interval time is T, then total sampling number $n = T/T_0$. So the prediction of v_{n+1} will be based on the $v_1, v_2, \ldots, v_n$.

Suppose for each prediction, we consider the historical information from *Tcurrent* – *T* to *Tcurrent*. During the time period *T*, we can collect speed information from sensors $v_1, v_2, \ldots, v_n$. Then we want to compute v_{n+1} based on the historical speed information and substitute $v(x_l, t)$ by v_{n+1} in equation (1).

$$v_{n+1} = f(v_1, v_2, \ldots, v_n)$$

that is:

$$v_{T/T0+1} = f(v_1, v_2, \ldots, v_{T/T0}) \qquad (2)$$

The following are several popular methods to predict v_{n+1} based on time series analysis [54, 55, 56, 57]:

3.2.1.1 Means of K-Nearest Neighbors

This method is a straightforward means of K- nearest neighbors:

$$v_{n+1} = \frac{1}{K} \sum_{i=n-K+1}^{n} v_i \qquad (3)$$

3.2.1.2 Exponential Smoothing Method

The exponential smoothing method is proven in [58] and applied in [54]. Suppose we already collect the speed data $v_1, v_2, \ldots, v_n$ from sensors, where v_n represents the average travel-speed during the last period of T_0 time units, v_{n-1} represents the average travel-speed during the second last period of T_0 time units, and so on. The one-step-ahead speed, namely the average travel-speed for the next T_0 time units, is predicted to be

$$v_{n+1} = \alpha \sum_{k=0}^{n-1} (1-\alpha)^k v_{n-k} + (1-\alpha)^n \frac{\sum_{k=0}^{n-1} v_{n-k}}{n} \tag{4}$$

where $0 \leq \alpha \leq 1$ is the smoothing constant. Selection of the proper smoothing constant α is an important problem. A widely used technique is introduced in [58].

In order to keep the predictions realistic, a maximum speed is used for each link and it can be determined by the speed limit on the street. If a predicted speed is higher than the maximum speed, it is revised down to the maximum speed.

3.2.1.3 Extrapolation through Several Values

We can use an extrapolation method, such as given v_n and v_{n-1}.

$$v_{n+1} = av_n + bv_{n-1} \tag{5}$$

where a, b can be constant or dynamic coefficients.

For example, we can use the form [55].

$$V_{n+1} = (3^{\alpha_n} - 1) V_n - V_{n-1} \tag{6}$$

where α is referred to as the local Hölder exponent [59]. It has the following form:

$$\alpha_n = \log_3 \left(\frac{V_{n-1} + V_n + V_{n+1}}{V_n} \right) \tag{7}$$

From this equation, the value of α for a point n (α_n) on the time series is based on speed values at the points n-1, n, and n+1. Speed at point n+1 (v_{n+1}) is unknown and in

practice, α_n must be predicted first before predicting v_{n+1}. This procedure includes three steps. Given a time series v_0 ,...,v_n:

(1) Compute the values of α_1,..., α_{n-1}.

(2) Predict the value of α_n. The nonlinear operation, α, is assumed to be a Markov chain and a method [59] based on conditional probabilities is used to predict the value of α_n. Since this method is beyong the scope of this thesis, we will not illustrate the details here.

(3) Perdition using $V_{n+1}=(3^{\alpha_n}-1)V_n-V_{n-1}$.

3.2.2 Machine Learning Models

Machine learning, a branch of artificial intelligence, is the study of computer algorithms that improve automatically through experience. These techniques have been applied in many fields. Also, machine learning has received much attention in the transportation area [60]. One of the most popular techniques in machine learning is Artificial Neural Network (ANN). ANN model has become one of the major machine learning approach to help improve transportation problems, such as travel time prediction [61, 62].

With this approach, we view the simulator as a nonlinear function of its n-parameter configuration and employ nonlinear regression to approximate it. We repeatedly use the sample data to train the ANNs to approximate the function. At each teaching (training) step, we obtain highly accurate error estimates of our approximation. We continue refining the approximation by training the ANNs further until error estimates become sufficiently small. Figure 28 shows the basic structure of an Artificial Neural Network. The input of this neural network is the average vehicle speed and traffic flow for each link, where the output is the predicted travel time. The hidden layers represent the non-linear regression process to approximate the relationship.

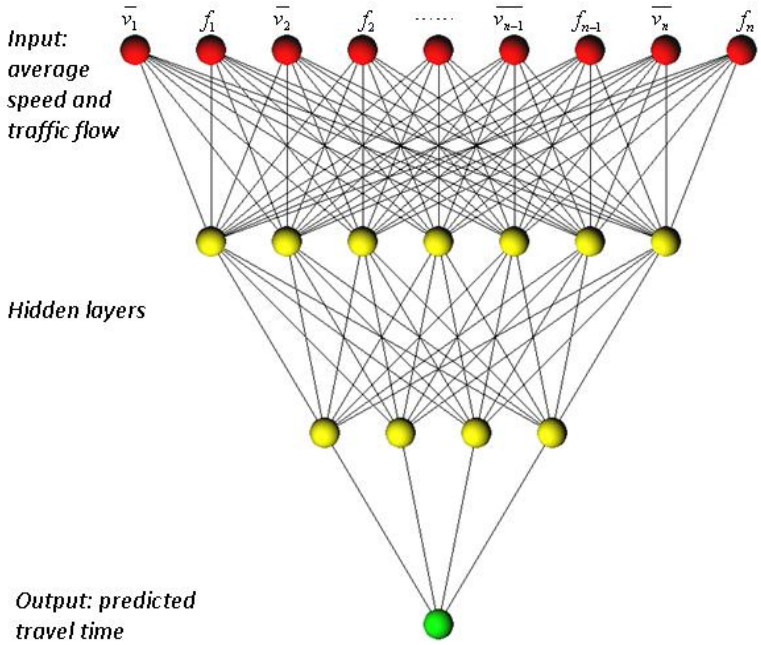


Figure 28. Basic Structure of ANN for Travel Time Prediction

Many research studies have used ANN techniques to predict travel-time that includes Advanced Neural Network [61] and Mix-structure Neural Network [62].

In Wei's study [61], ANN has been shown to be an effective approach for travel time prediction. Also, the experimental results in [64] showed that the expected travel time prediction accuracy with ANN is approximately 96%, which can be considered as high prediction accuracy in most cases.

The Mixed-structure NN model has the capacity to use the data from detector segments to predict travel time of non-detector segments [63]. Therefore, we can reduce the number of detectors by the improvement of travel time prediction technology.

Other machine learning techniques used in travel time prediction include support vector machine (SVM), Bayesian and K-Nearest Neighbors (KNN). Bayesian and KNN are basic machine learning approaches that must usually be combined with other methods such as ANN to obtain accurate prediction results. For SVM, it is important but very

difficult to select proper kernel functions and optimal parameters. Theoretically, ANN can capture any relationship between output and input values, however, it can suffer from over-fitting, and prediction accuracy can also be improved.

3.3 Boosting Neural Network for Travel Time Prediction

In this section, we will present our travel time prediction algorithm based on a boosting neural network. This algorithm contains two parts: the preliminary and the prediction components. We will introduce the framework of the prediction part first, and then give an overview of the preliminary step. In later subsections, details of the prediction algorithm and the preliminary step are presented.

3.3.1 Algorithm Framework

Boosting is a general and provably effective approach of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb [45]. Boosting has been shown to be effective for classification [45, 48, 51, 52] and regression [46, 47, 52]. Travel time prediction can be considered as a classical regression problem. Therefore, we propose the boosting-based travel time prediction algorithm. In this thesis, our model is based on AdaBoost [45], but we modify specific steps of the existing boosting algorithms to suit our regression problem in traffic prediction.

We begin with a neural network "basic" learner that is responsible for finding the rough rules of thumb. It then repeatedly applies a different weight distribution over the training data for attributes such as traffic flow and average speed [43]. Each time it is called, the basic learning algorithm generates a new weak prediction rule, and after many rounds, the boosting algorithm combines these weak rules into a single prediction rule that will be more accurate than the weak rules. The framework of the proposed prediction algorithm is described as follows:

Input: a basic learner called BaseLearn (a feed-forward neural network is used here), an integer T specifying the number of iterations, and N labeled training data samples in the format of time series. Each data sample is a vector (X_i, T_i, C_i), and X_i is also a vector including collected average speed and traffic flow for each link of a pre-defined path P_a during the collection period (t_a, t_b). T_i is the actual travel time for the travel on the path P_a whose travel beginning time is $t_b + \Delta t$. During the each iteration of training, predicted value $T_t^{(p)}(X_i)$ will be compared with T_i, where t is the current iteration number. C_i is the cost used for measuring the importance of each sample.

Output: a strong learner with improved prediction accuracy.

1. Initialize the weight distribution of all the data samples in the training set: $D_1(i) = \frac{1}{N}$

where *N* is the total number of data samples in the training set.

2. Do for iteration $t = 1, 2, ..., T$

2.1 Call BaseLearn using distribution D_t on the training set, returning a weak hypothesis h_t.

2.2 Evaluate the error (weight) $r_t(i)$ for every data sample i in the training set using the weak hypothesis h_t.

2.3 Compute the error ε_t of the weak hypothesis h_t.

2.4 Choose a_t to measure the weight of the weak hypothesis h_t.

2.5 Update the weight distribution of all the samples in the training set, returning D_{t+1}.

3. Output the final strong hypothesis by combining the weak hypotheses:

$\{h_t, t = 1, 2, ...T\} \rightarrow H$

There are two types of weights in the algorithm: one for each data sample in the training set and one for the hypothesis. Boosting maintains a weight $r(i)$ for each sample i in the training set. The higher the weight $r(i)$, the more the instance i influences the next learned hypothesis. For each iteration, the weights are adjusted to reflect the performance of the corresponding hypothesis, with the result that the weight of hard instances is increased. For the weight of the hypothesis, higher weight should be distributed to the hypothesis which has higher prediction accuracy, so the hypothesis can have increased influence on the final strong hypothesis. The detailed approach of weighting training data samples and hypotheses will be illustrated in subsection B.

Traffic data such as vehicle speed information is necessary to drive the algorithm. The frequency of traffic data collection influences the consumption of bandwidth, which is a limited and expensive resource in vehicle networks. Therefore, finding the lower bound of the data collection frequency is desirable to minimize bandwidth requirements while maintaining accurate travel time predictions. This step is the preliminary part for the boosting neural network approach. After determining the lower bound of the data collection frequency, we can use this lower bound to obtain the input for the neural network. The framework of this preliminary step is shown below:

Input: Raw traffic data such as average vehicle speed for each link collected at some rate. The value of the rate should be very small, such as several seconds, which ensures a sufficient target range for finding the lower bound of the collection frequency.

Output: The lower bound of the data collection frequency while maintaining high prediction accuracy.

1. Initialize the data collection frequency f_0 and total sampling number n.

2. Do for iteration $t = 1, 2, ..., T$

2.1 Predict travel time Y_T based on the boosting neural network approach.

2.2 Compute prediction error ε_i using the current data collection frequency f_{0i}.

2.3 Compare the prediction error ε_i with ε_{i-1}, ε_{i-2}, …, ε_{i-k} and determine if the stopping criterion is met. If it is, then we determine the approximate target range.

3. Search further in the approximate target range and obtain the lower bound on the data collection frequency.

3.3.2 Prediction Algorithm Details

The prediction algorithm is composed of three major steps: initialization, computation and combination. The computation step includes several important minor steps, such as evaluating error for each data sample, measuring the weight of the weak hypothesis and updating the weight distribution of all the samples. The details for each step of the prediction algorithm will be illustrated as follows.

1) Determination of the cost for each sample (Preparing for input data): The cost items are used to measure the importance of samples [48]. Suppose the historical data collection period or monitoring interval is (t_a, t_b). The data sample collected around t_b should be given more cost than that collected around t_a, since the former one has more influence on the prediction result. For example, if the collection period is from 3pm to 4pm, and we would like to predict the total time of travel whose beginning time is 4:10pm, then the data collected at 3:50pm should be given higher cost than that collected at 3:10pm. It is computed as:

$$C_i = \frac{t_b - t_a}{\sum_{j=1}^{N}(t_j - t_a)} \times \frac{t_i - t_a}{t_b - t_a} \tag{8}$$

where t_i is the collection time of sample i.

2) Initialization (Step 1): One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on the data sample i in round t is denoted $D_t(i)$. Initially, all weights are set equally, but on each round, the weights of the examples with large errors in the later iteration are increased so that the basic learner is forced to focus on the hard examples in the training set.

3) The basic learner (Step 2.1): The proposed boosting algorithm can use any basic learner. In this thesis, we utilize a feed-forward neural network (FFNN) [49], which has the same structure as that in Figure 24. The input to the feed-forward neural network is the average speed and traffic flow measured for each link, where $\overline{v_j}$ denotes the input vector of average speed collected during a pre-defined period for link j, and f_j denotes the input vector of traffic flow collected during the same period for link j.The output of the neural network is the predicted travel time.

4) Evaluation of error (weight) for each data sample (Step 2.2): The error of each data sample is different from the cost of each sample. The cost is a constant value which reflects the uneven identification importance among samples, while the error is a variable changed each iteration to reflect how well the weak hypothesis makes prediction for this sample. The higher the difference between prediction value and actual value, the more error will be distributed to this sample, which forces the weak hypothesis to focus on the samples with large error called hard samples in the next iteration. The simplest form to evaluate the error is the linear form [46].

$$r_t(i) = \frac{|T_t^{(p)}(X_i) - T_i|}{S} \tag{9}$$

where $r_t(i)$ is the error of data sample i in the training set in the t^{th} iteration, $S = \sup|T_t^{(p)}(X_i) - T_i|, i = 1,2,...N$, T_i is the actual output for the i^{th} example and $T_t^{(p)}(X_i)$ is the predicted value of T_i in the t^{th} iteration.

Here, we adopt another form called the saturated law, which is expected to be more accurate than the linear form because it is less prone to over-fitting.

$$r_t(i) = 1 - \exp(-\frac{|T_t^{(p)}(X_i) - T_i|}{S}) \tag{10}$$

5) Computation of the error for each weak hypothesis (Step 2.3): The training error ε_i of the basic regressor h_i is determined by the weight and error of each example in the training set. It can be computed as:

$$\varepsilon_t = \sum_{t=1}^{N} D_t(i) \cdot r_t(i) \tag{11}$$

6) Computation of the weight for each weak hypothesis (Step 2.4): The weight of weak hypothesis measures its influence on the final strong hypothesis. The weak hypothesis having higher prediction accuracy should be given higher weight, so this weak hypothesis can have increased influence on the final strong predictor. The weight of each weak hypothesis in the standard AdaBoost [45] is computed as:

$$a_t = \frac{1}{2} \cdot \ln(\frac{1-\varepsilon_t}{\varepsilon_t}) \tag{12}$$

However, the method of weighting hypotheses should be adjusted for different application environments. For example, since we consider the factor of cost C_i for each sample i, the parameter C_i should be added to our weighting method. Yanmin Sun [48] proposed a boosting algorithm called AdaC2, where he computed the weight of each hypothesis as follows:

$$a_t = \frac{1}{2} \log \frac{\sum_{i, T_t^{(p)}(X_i) = T_i} C_i \cdot D_t(i)}{\sum_{i, T_t^{(p)}(X_i) \neq T_i} C_i \cdot D_t(i)} \tag{13}$$

However, this approach is applied to the classification problem. For our regression problem, considering only whether the computation result is the same as the actual value is not sufficient. We should know the size of the difference between the computated result and actual value. So the following method is used to measure the difference:

$$If \cdot r_t(i) = 1 - \exp(-\frac{|T_t^{(p)}(X_i) - T_i|}{S}) \text{ is smaller } (l\arg er) \text{ than } \eta,$$

then sample *i* is counted in the numerator (denominator)

where η is a threshold.

Therefore, the approach for weighting hypothesis in our work is as follows:

$$a_t = \frac{1}{2}\log\frac{\sum_{i, r_t(i)<\eta} C_i \cdot D_t(i)}{\sum_{i, r_t(i)\geq\eta} C_i \cdot D_t(i)} \tag{14}$$

7) Updates of weight distribution (Step 2.5): There are many methods to update the weight distribution. Some extra parameter can be introduced to improve the prediction performance [46], but how to select a proper parameter is difficult. In H. Drucker's work, the following method is proposed for the regression problem.

$$D_{t+1}(i) = \frac{D_t(i) \cdot a_t^{(r_t(i)-1)}}{Z_t} \tag{15}$$

where $Z_t = \sum_t D_t(i) \cdot a_t^{(r_t(i)-1)}$ is a normalization factor.

To address the above issues, we propose the following approach of weight distribution updating:

$$D_{t+1}(i) = \frac{C_i \cdot D_t(i) \cdot a_t^{(r_t(i)-1)}}{Z_t} \tag{16}$$

where $Z_t = \sum_t C_i \cdot D_t(i) \cdot a_t^{(r_t(i)-1)}$ is a normalization factor.

8) Combination of weak hypotheses to a strong predictor (Step3): One way to obtain the final strong hypothesis is to use the weighted average of weak hypotheses. Although this approach has some advantages in practice, it is not a natural extension of AdaBoost [50]. So we adopt the weighted median method which extends AdaBoost natually.

$$\beta_t = \varepsilon_t / (1-\varepsilon_t)$$
$$h_f(X_i) = \inf\left\{ Y \in R : \sum_{t:h_t(X_i)\le Y} \log(1/\beta_t) \ge \frac{1}{2} \cdot \sum_{t=1}^{T} \log(1/\beta_t) \right\} \quad (17)$$

9) Choice of the number of iterations (Error Upper Bound of the Final Hypothesis): The stopping criterion of the proposed algorithm is very important. It can be determined by examining the error of the weak hypothesis, and checking if the error is reduced to an acceptable value. Usually for AdaBoost [45], if ε_t is larger than ½, then the loop should be terminated. In order to avoid slow convergence, we also set the maximum iteration number $T_{\max}$. So on each round, after calculating the error of the weak hypothesis; we must check if the following stopping criterion is met.

$$If\ \varepsilon_t > \frac{1}{2}\ or\ T \ge T_{\max},\ then\ T = t-1$$

and abort loop

After this criterion is met, the weak hypotheses should be combined to a strong hypothesis, whose prediction accuracy can be expected to improve compared to the weak learners.

3.3.3 Preliminary Step Details

Traffic data collection and transmission consume bandwidth resources. It is not unreasonable to expect that increased data collection frequency, and thus increased bandwidth consumption will result in more accurate prediction results. But bandwidth is a limited and expensive resource in networks, especially wireless networks. Hence it is important to find the lower bound of the data collection frequency which minimizes bandwidth requirements while maintaining high prediction accuracy. We compute the lower bound of the data collection frequency as the preliminary step with a binary search-like approach, and then apply the lower bound to obtain the input for the boosting neural network approach. The details are presented as follows:

1. **Choose** the monitoring interval T and the range of the data collection period T_0 based on empirical statistics. For example, choosing T to be 30 minutes means the travel time prediction for the travel whose beginning time is 4:00 pm is based on traffic data collected from 3:30 pm to 4:00 pm. T_0 is typically chosen to be 1, 3, or 5 minutes. So the range of T_0 can be chosen from 10 minutes to 1 second ($T_{0\max}$ = 10 minutes). Then we use the following steps to decide the appropriate value.

2. **Compute** the initialized data collection frequency $f_{0\min} = 1/T_{0\max}$ and total sampling number $n = T/T_{0\max}$. The travel time prediction will be based on the raw traffic data from time step t_1 to t_n.

3. **Predict** travel time using the boosting feed-forward neural network approach illustrated in the previous section.

4. **Compute** the prediction error using this data collection frequency. There are several methods to compute the prediction error, such as using the following equation to compute Percentage Prediction Error (PPE).

$$\varepsilon\left(t, T^{PRED}(t)\right) = \frac{\left|T(t) - T^{PRED}(t)\right|}{T(t)} \tag{18}$$

where $T(t)$ denotes the actual travel time, $T^{PRED}(t)$ denotes the predicted travel time computed with the boosting-based approach, and t denotes the beginning time of a travel that we make travel time prediction for . Now we can get one pair of data collection frequency and prediction error $(f_{0\min}, \varepsilon)$.

5. Use a binary search approach to find the minimal data collection frequency. For example, $T_{0\max}/2$ is used as the data collection period for the next step. Then the sampling number n is changed to $2T/T_{0\max}$. So the prediction will be based on $v_1, v_2, ..., v_{2T/T_{0\max}}$. Repeat steps 3 and 4. After that new prediction error is computed and compared with that of the last iteration. If the prediction error is less than that of the last time, use $T_{0\max}/4$ for the next iteration.

6. Determine the approximate target range. Repeat step 5 until we find the prediction error change becomes sufficiently small. For instance, if the prediction error is the same for update periods less than $T_{0\max}/2^m$ ($T_{0\max}/2^{m+1}$, $T_{0\max}/2^{m+2}$, …), we can focus on the target range $(T_{0\max}/2^m, T_{0\max}/2^{m+1})$.

7. Search further in the range $(T_{0\max}/2^m, T_{0\max}/2^{m+1})$ to find the required data collection period T_{0HB} and thus f_{0LB} (the lower bound of the data collection frequency which minimizes bandwidth requirements while maintaining high prediction accuracy). Repeat the above steps until the difference of prediction errors for two adjacent iterations is small enough to meet the requirement:

$$0 \le \varepsilon_t - \varepsilon_{t+1} < \xi \tag{20}$$

where ε_t is the prediction error of the t^{th} iteration, and ξ is the a threshold, which is between 0 and 1, such as 0.001.

3.4 Experiments and Results

To evaluate the boosting-based neural network approach, we conducted several experiments. The experiment environment and data collection will be presented first, followed by the experimental results.

We conducted experiments using simulated data from the microscopic simulator VISSIM. The simulated traffic network has 10*10 intersections. Two data sets are collected: the average speed and traffic flow for each link (with a collection interval of 1 second), and the travel time through a selected path. The first data set is the input to the prediction algorithm. After predicting the travel time, the output of the prediction algorithm (the predicted travel time) is compared with the actual travel time (second data set) and the prediction accuracy is the ratio between the predicted time and the actual time.

In order to evaluate the effectiveness of the proposed algorithm, we conducted three kinds of evaluation: comparisons of accuracy using different prediction methods, comparisons of prediction accuracy using different numbers of iterations of the boosting approach, and comparisons of prediction accuracy under different data collection frequencies.

3.4.1 Comparisons of Different Prediction Methods

We conduct comparative experiments of three different prediction approaches. One is our boosting feed-forward neural network approach (BFFNN), the second is feed-forward neural network method (FFNN), and the third is an historical mean prediction approach (HMP). FFNN uses neural networks to predict travel time without boosting. HMP method uses the average travel time of the historical traffic data. Here, we conduct 20 simulation runs (replications) to collect the historical data. The experimental results for these three prediction methods are shown in Figure 29.

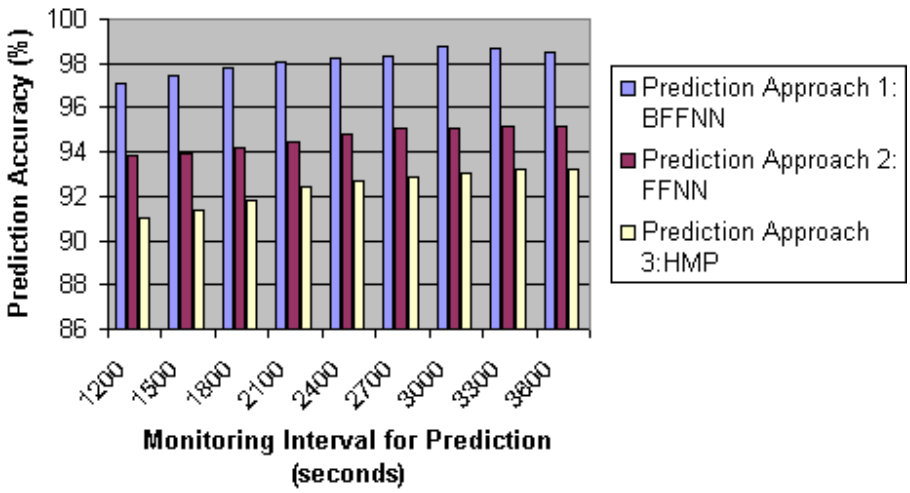


Figure 29. Comparisons of Different Prediction Approaches

The results in Figure 29 show the prediction is more accurate as the monitoring interval time is increased for all three approaches. The prediction accuracy of BFFNN is higher than FFNN and HMP across the different monitoring interval times used for these experiments. The boosting neural network algorithm improves prediction accuracy by approximately six percent over the historical mean and approximately three to four percent over the forward-feed neural network.

3.4.2 Comparisons of Different Iteration Number of Boosting

The number of iterations of boosting has an important influence on the prediction accuracy. The prediction error should decrease with a large number of iterations. The experimental results with different number of iterations are shown in Figure 30.

As shown in Figure 30, ten iterations are not sufficient to obtain high prediction accuracy. Iterating 30 times improves travel time prediction by approximately two percent. If the number of iterations is 50, the prediction accuracy increases somewhat, but it requires a longer running time and consumes more resources than that with 30 iterations.

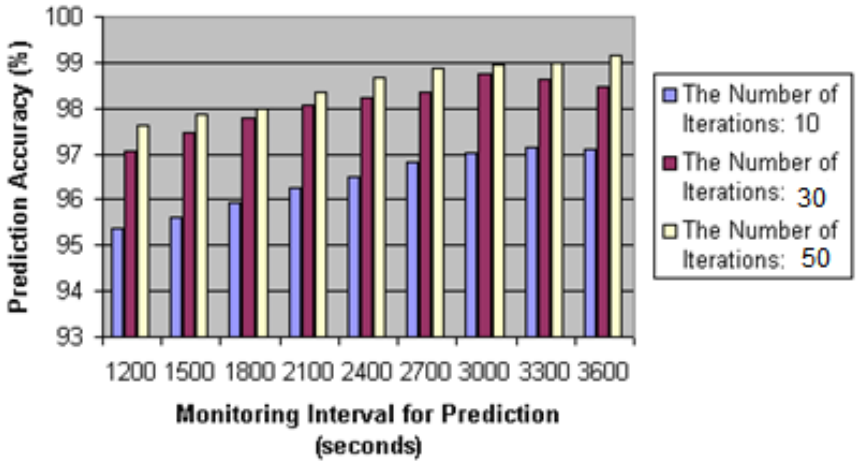


Figure 30. Comparisons of Different Iteration Number

3.4.3 Comparisons of Different Data Updating Frequency

In our experiments, we initialize the minimum data collection frequency. We then double the collection frequency, and if the prediction accuracy becomes higher than before, we change the collection frequency to the new value. We repeat this process, and obtain the lower bound of the data collection frequency, while maintaining accurate travel time predictions. The experimental results are shown in Figure 31 and Figure 32.

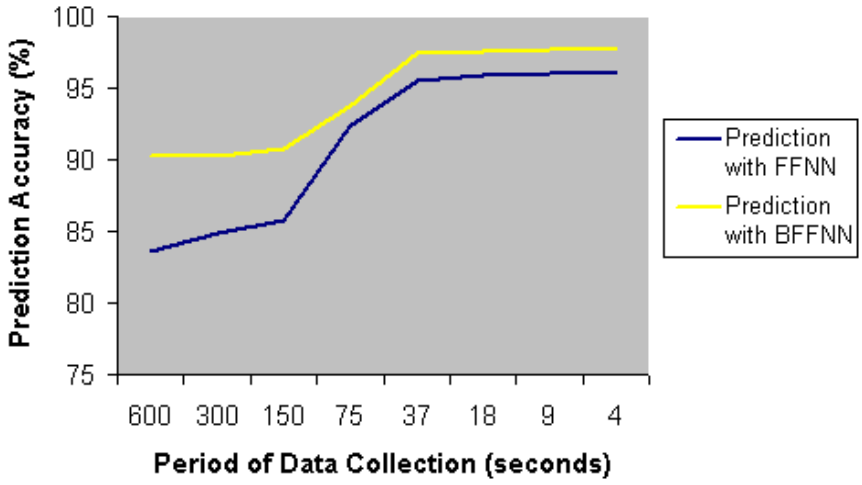


Figure 31. Comparisons of Different Data Updating Frequency

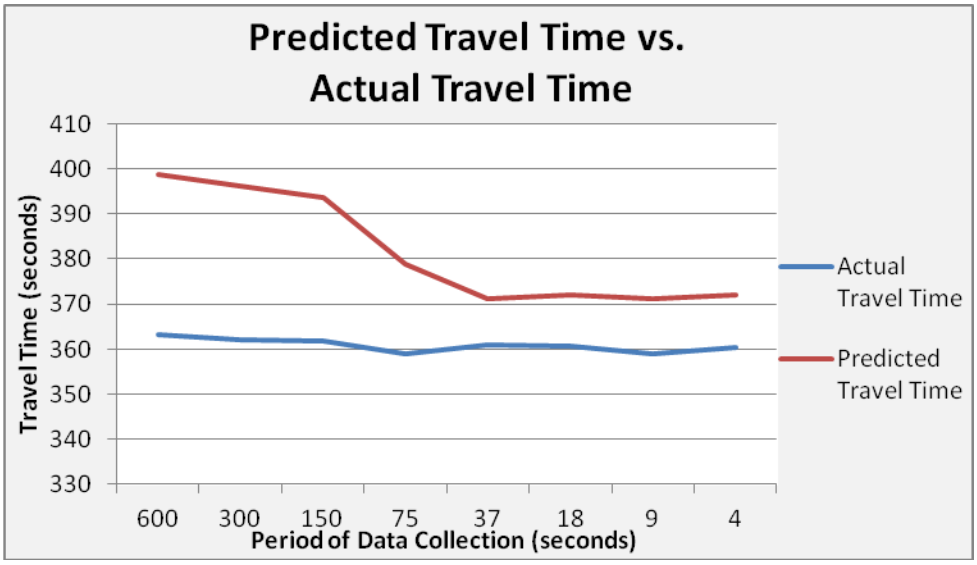
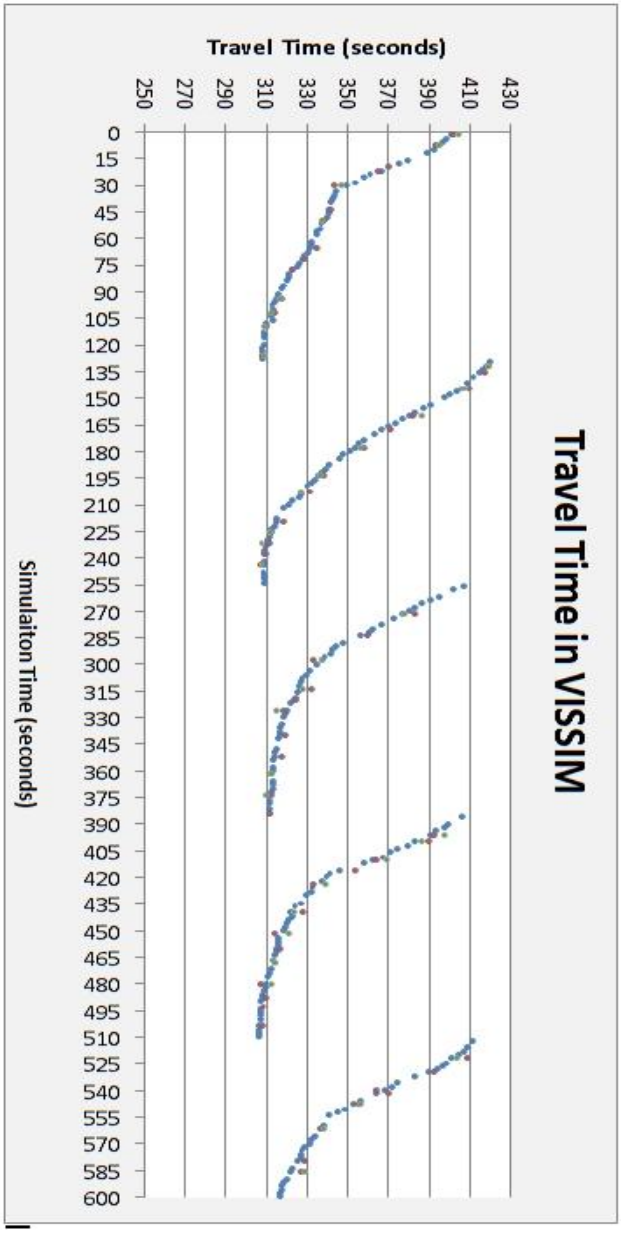


Figure 32. Predicted Travel Time vs. Actual Travel Time

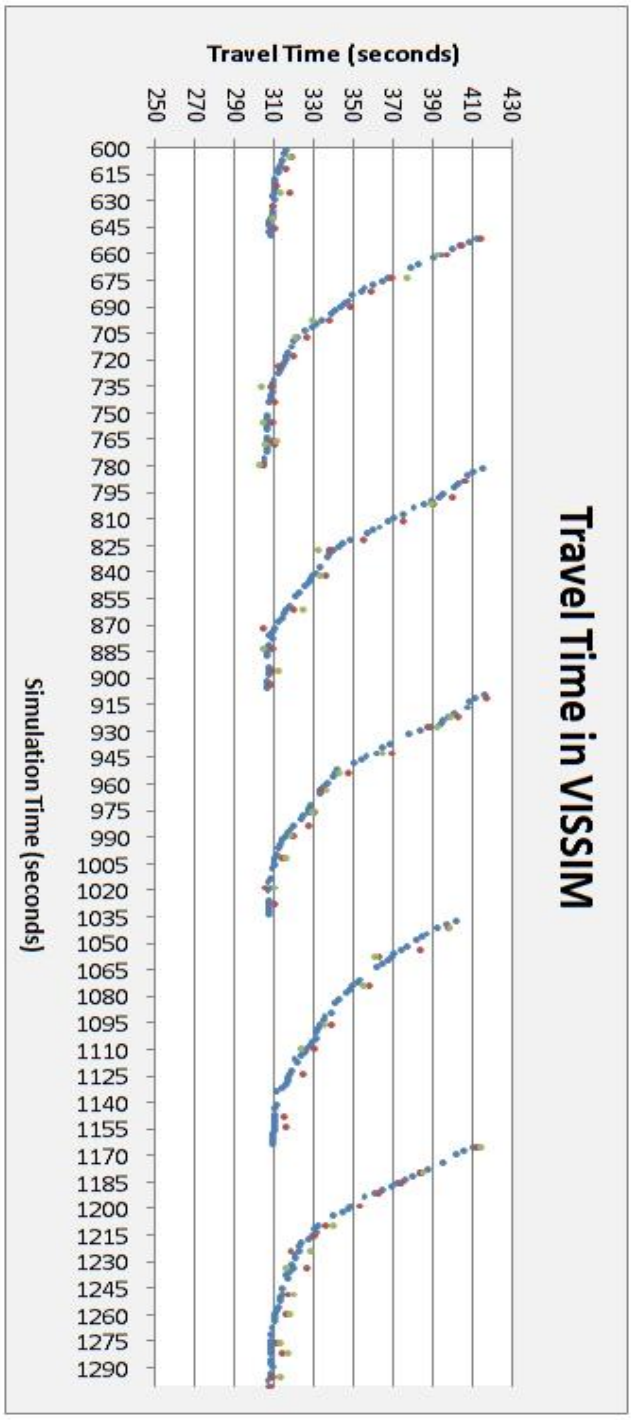
The results show the following trends. First, BFFNN has higher prediction accuracy than FFNN for any of the tested updating periods. This demonstrates the improved accuracy of the boosting approach; it can increase the prediction accuracy of the basic neural network approach. Second, the optimal data updating period is approximately 37 seconds for both of the algorithms, which shows for this network example, we should collect data about every 37 seconds to maintain high prediction accuracy while minimizing the use of bandwidth resources.

The result also shows the impact of signal timing on performance. When the data collection period is less than around 120 seconds (which is the cycle time of traffic controllers), the prediction accuracy begins to increase. The reason is because we have collected data samples for different phases of traffic controllers with a 120-second collection period, including both green phase and red phase, and these data samples are relatively sufficient to make predictions. When the data collection period is less than around 30 seconds, the prediction accuracy is high, since the data samples include more data for different phases of traffic controllers, which makes the prediction more accurate.

In order to explore the impact of signal timing further, we collect data from VISSIM and examine the trend of actual travel time related to signal timing. In Figure 33, the travel time shows some cycles and the cycle is around 130 seconds, which is highly similar as the cycle of traffic controllers. In each cycle, travel time decreases to the trough gradually and then jump to the crest at the beginning of next cycle. This is because in the red light phase, travel time is relatively high and when it turns into the green light phase, travel time decreases gradually until the next red light phase comes.



(a)

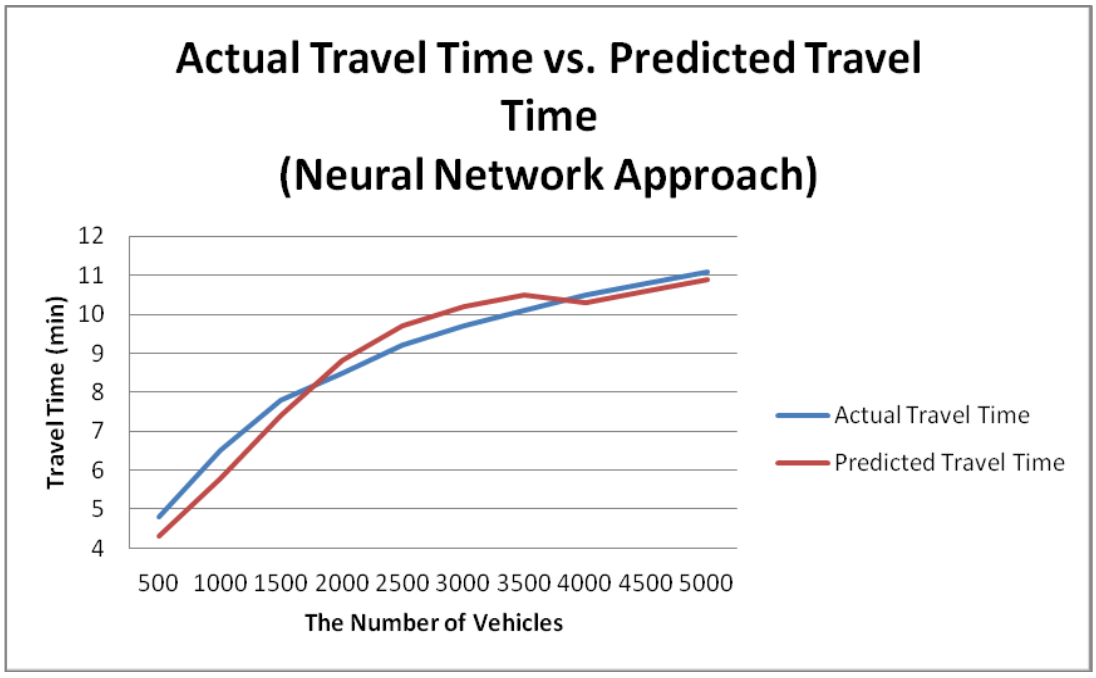


(b)

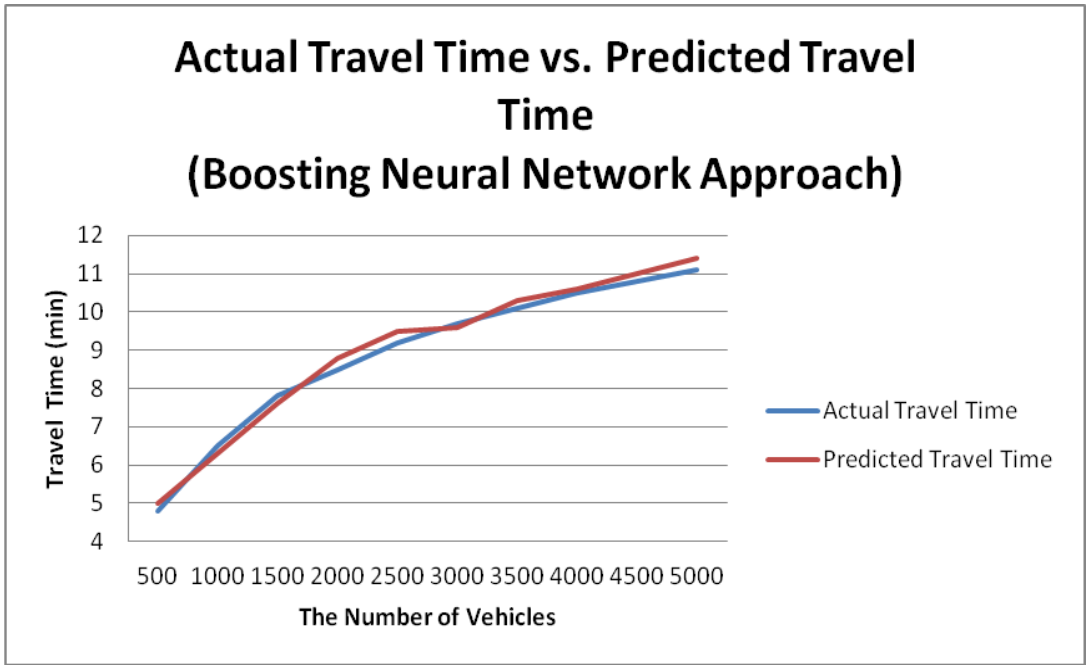
Figure 33. Travel Time in VISSIM

3.4.4 Comparisons of Different Vehicle Numbers

In the real world, rush hour or non-rush hour will have a great impact on travel time. In congested situations, vehicles will have a longer travel time to arrive at the destination. We use different vehicle numbers in VISSIM to simulate rush hour and non-rush hour and record both the actual travel time and predicted travel time. Two prediction approaches are evaluated: neural network approach and boosting neural network approach. The results are shown in Figure 34. From the figure, it is obvious to see that after boosting, the prediction accuracy is much higher than that of the neural network without boosting approach. Another trend is that as the number of vehicles increases, the travel time increases also, which indicates that vehicles will take longer time to arrive at the destination in the congested scenario.



(a)



(b)

Figure 34. Comparisons of Different Vehicle Numbers

3.5 Discussion - Basic Learner Setting

Theoretically, the proposed boosting algorithm can use any basic learner. Here, we utilize a feed-forward neural network (FFNN). We are interested in exploring the impact of different neural network settings on performance.

3.5.1 Neural Network Structure

We vary the number of hidden layers and the number of neurons to identify the appropriate FFNN structure, which can provide relatively good prediction accuracy before the boosting procedure. We use cross validation (which will be explained in detail in the section for avoiding overfitting) to do the model selection. Our neural network is first run with a single hidden layer and the number of neurons in the hidden layer is varied from 3 to 30. The best number of hidden neurons is decided based on the

performance of the proposed models. The prediction results are shown in the following figure in which Percentage Prediction Error (PPE) (in equation 18) is shown on the Y-axis.

Figure 35 shows the percentage prediction error with different numbers of neurons. The FFNN model with 15 neurons for the hidden layer gives the best results in the experiment. As the number of neurons increases from 3 to 15, the PPE decreases and its value is 6.1% with 15 neurons. The low prediction accuracy with low number of neurons indicates that a too simple model won't learn the specificities of the data well. When the number of neurons reaches 20, the PPE increases dramatically, which indicates that a too complex model will learn irrelevant details of the data and eventually its noise [84] (which may cause overfitting, we will discuss this issue in detail later). From this experiment, 15 is a good choice for the number of neurons for one hidden layer case.

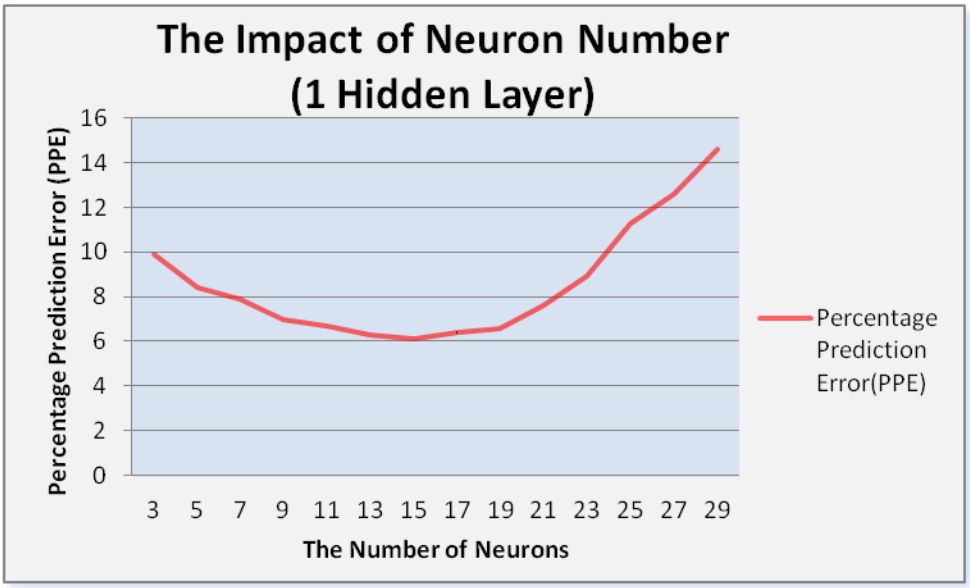


Figure 35. Different Neuron Numbers for Single Hidden Layer

The FFNN with a single hidden layer may not be a satisfactory universal function approximator in practice. This is particularly true when the function approximated is highly nonlinear and multidimensional, which are typical of the problems experienced in

many transportation applications [81]. It has been found in practice that FFNN capacity increases as the number of hidden layers increases. Therefore, the FFNN architecture is also run using two hidden layers and we vary the number of neurons for the two hidden layers from 3 to 15 and examine their performance.

Figure 36 shows the percentage prediction error of different numbers of neurons in two hidden layers. When we fix the number of neurons of one hidden layer and increase the number in another layer, the percentage prediction error decreases first and then after some value, the percentage prediction error starts to increase. This trend is similar to the case for a single hidden layer, since the neural network model cannot be too simple or too complex to get high prediction accuracy. A too simple model won't learn the specificities of the data well and will result in low prediction accuracy. A too complex model will learn irrelevant details of the data and eventually its noise [84]. The best FFNN with two hidden layers has five neurons in the first hidden layer and seven neurons in the second hidden layer (the point pointed by red arrow in Figure 36). As expected, the two hidden layer FFNN gives better results when compared with the single layer FFNN. The best FFNN with single layer has 15 neurons and the percentage prediction error is 6.1%, while the best FFNN with two hidden layers has percentage prediction error 4.2%.

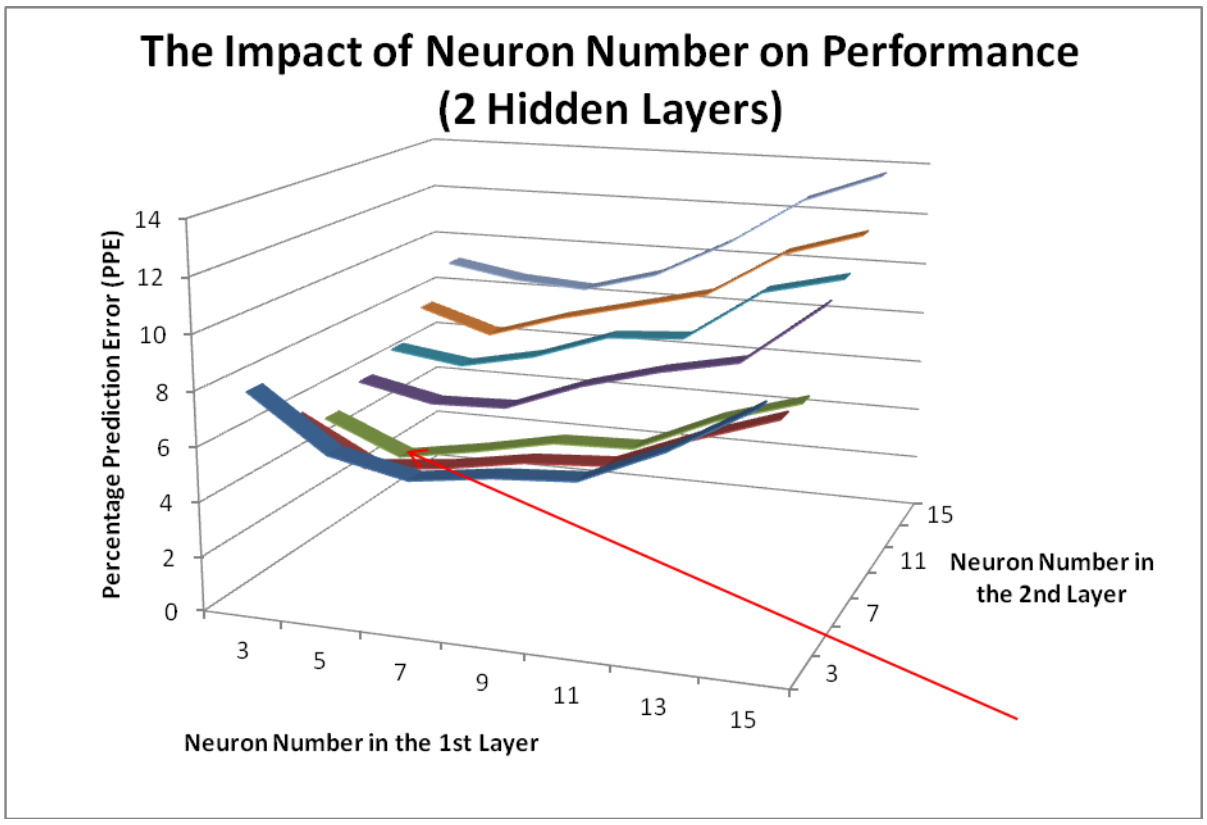


Figure 36. Different Neuron Numbers for Two-Hidden Layer

If more than two hidden layers are incorporated into the model, we may obtain better FFNN capacity and more accurate prediction accuracy; however, training such a FFNN may require significantly increased effort and time for selecting the best architecture [81]. Therefore, two hidden layer FFNN is a good choice for our model.

3.5.2 Stopping Criteria

There is an important question of how many iterations are required to train a neural network. There are mainly two paradigms, late stopping and early stopping. Late stopping means that the network is trained until a minimum error on the training set is reached (the network is clearly overfitted). Then different techniques are used to exterminate nodes in the network (known as pruning). By doing so eventually a good generalization ability is reached [82]. There are many different pruning algorithms, but

this is beyond the scope of our discussion. In our work, we use late stopping first to see when a minimum error on the training set is reached, and then we will use early stopping to avoid overfitting. Figure 37 shows the impact of iteration number on training error for both best FFNN with one hidden layer and two hidden layers. The results show the following trends. First, with the same iteration number, the best FFNN with two hidden layers has higher prediction accuracy than that with one hidden layer. Second, the best FFNN with two hidden layers uses fewer iterations to reach the minimum learning error. After 2000 iterations, the training error of the best FFNN with a single hidden layer remains the same; while the best FFNN with two hidden layers only uses 1600 iterations to reach the minimum training error.

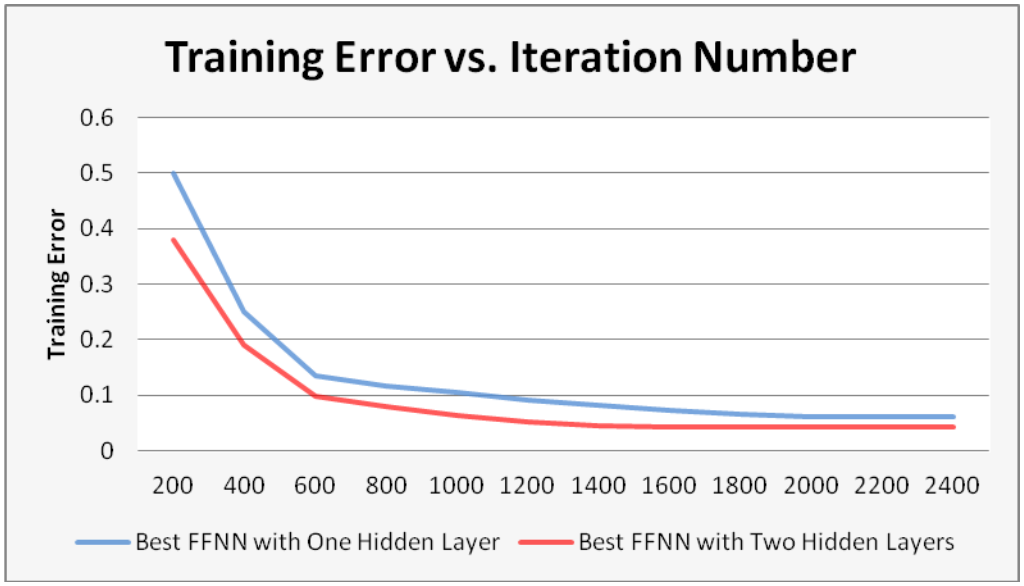


Figure 37. The Impact of Different Neuron Numbers for Two-Hidden Layer Case

Compared to late stopping, early stopping is a way to avoid overfitting. The learning process with the early stopping method is monitored all the time and training is terminated as soon as signs of overfitting appear [82]. In early stopping, the training set is split into a new training set and a validation set. The validation error rate is computed periodically during training and we should stop training when the validation error rate

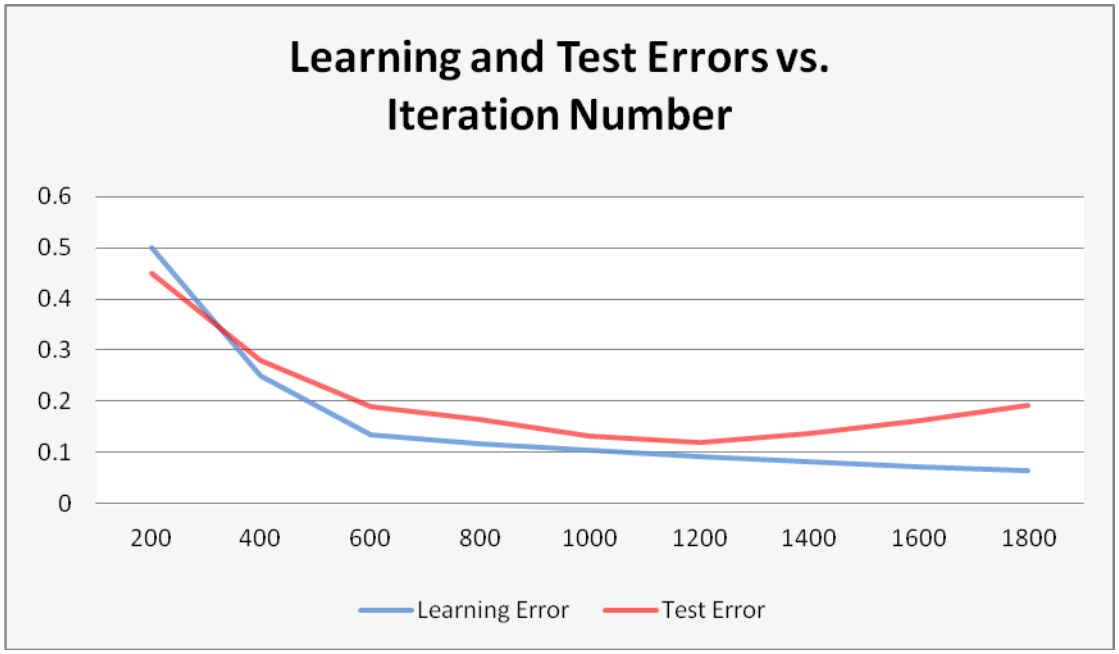
starts to increase. The results for early stopping will be presented in the next section regarding "approaches to avoid overfitting".

3.5.3 Overfitting and Regularization

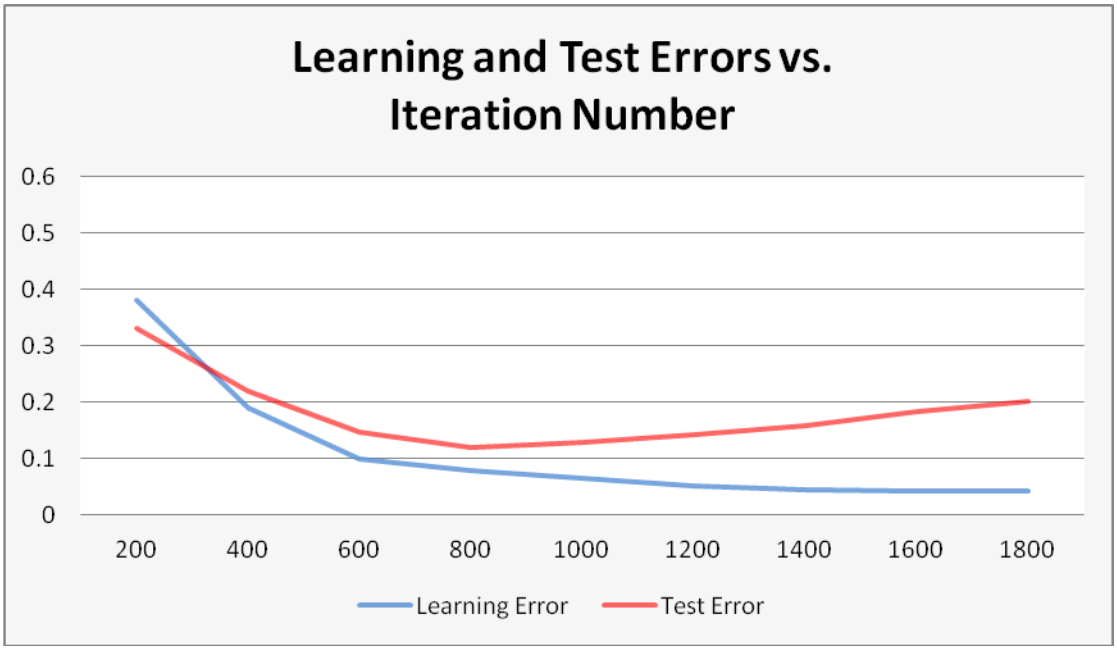
Overfitting occurs when a model begins to memorize training data rather than learning to generalize from trend. Such a model will typically fail when making predictions about new data, since the model has not learned to generalize [83]. There are different approaches to avoid overfitting and we will evaluate the following methods: 1) early stopping; 2) adding L1 norm; 3) adding L2 norm; 4) cross validation.

3.5.3.1 Early Stopping

We split the original training set into a new training set (80% percent of the original training set) and a validation set (20% percent of the original training set) and monitor the performance of the validation set. Figure 38 shows the trend of learning error and test error as the iteration number increases. Learning and test errors for the best FFNN with single hidden layer is shown in Figure 38(a) while Figure 38(b) presents the results for the best FFNN with two hidden layers. In Figure 38(a), the test error starts to go up after 1200 iterations while the learning error still decreases at that time. This indicates that we should stop training before 1200 iterations to avoid overfitting.



(a)



(b)

Figure 38. Learning Error and Test Error

In Figure 38(b), the results show the following trends. First, the learning error keeps increasing until after approximately 1600 iterations while the test error begins to

increase after approximately 800 iterations. The algorithm should halt when the performance with the validation test stops improving, so in order to avoid overfitting, we should stop training before 800 iterations. Second, it takes fewer iterations for FFNN with two hidden layers to reach the point for early stopping than FFNN with a single hidden layer, which means we should stop earlier if we use two hidden layer neural network model.

3.5.3.2 L1Norm and L2 Norm

In this section, we compare two different regularization methods for preventing overfitting: L1 regularization and L2 regularization. The first, L1 regularization, uses a penalty term which encourages the sum of the absolute values of the parameters to be small. The second, L2 regularization, encourages the sum of the squares of the parameters to be small. Besides comparing L1 and L2, we also use cross validation to determine optimal regularization parameters.

Figure 39 shows the test error of different regularization parameters for L1 and L2 in the best FFNN model with two hidden layers. There are several trends in this figure. First, L2 norm has a lower test error than that of L1 norm for our data, regardless of the regularization parameter. Second, both L1 and L2 yield good recommendations for the value of the regularization parameter. The optimal regularization parameter is around 0.001 for L2 norm while the optimal regularization parameter is around 0.01 for L1 norm.

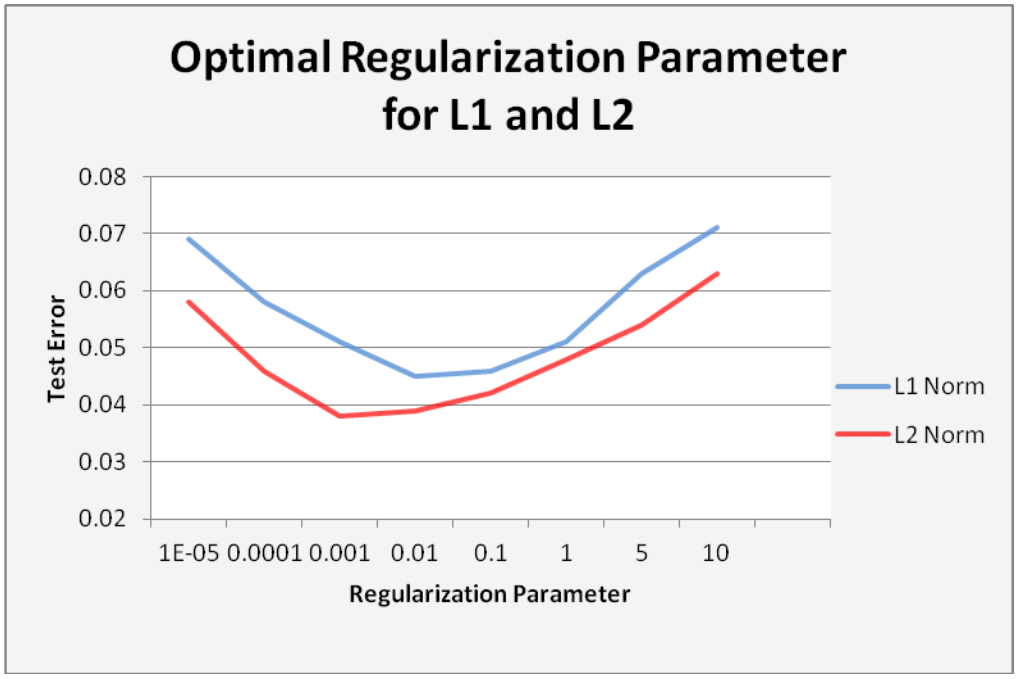


Figure 39. Optimal Regularization Parameter for L1 and L2

3.5.3.3 Cross Validation

Besides regularization, there are other algorithms to avoid overfitting. The simplest algorithm is the Test-Set Method [84]: we divide the data into a Learning Set (large) and a Test Set (smaller). We train our neural network on the Learning Set and measure the error made predicting on the Test Set. This technique, however, cannot be applied when the amount of data is small because then we need the whole data set for training. Moreover, if the data set is small (which is often the case), the generalization error depends heavily on the choice of the Learning and Test sets [84].

We can use cross validation to prevent overfitting in our training. The training sample is split into multiple subsets or folds, for example K folds. For each fold, we use it as Test Set and use the whole data set except it as Learning Set. We are interested in finding an appropriate K for training, so we examine the cross validation error with different K values. The result in Figure 40 shows that K between 4 and 10 is a good

trade-off. If K is chosen to be less than 4, the cross validation error is too large; If K is larger than 10, it makes no difference for our results.

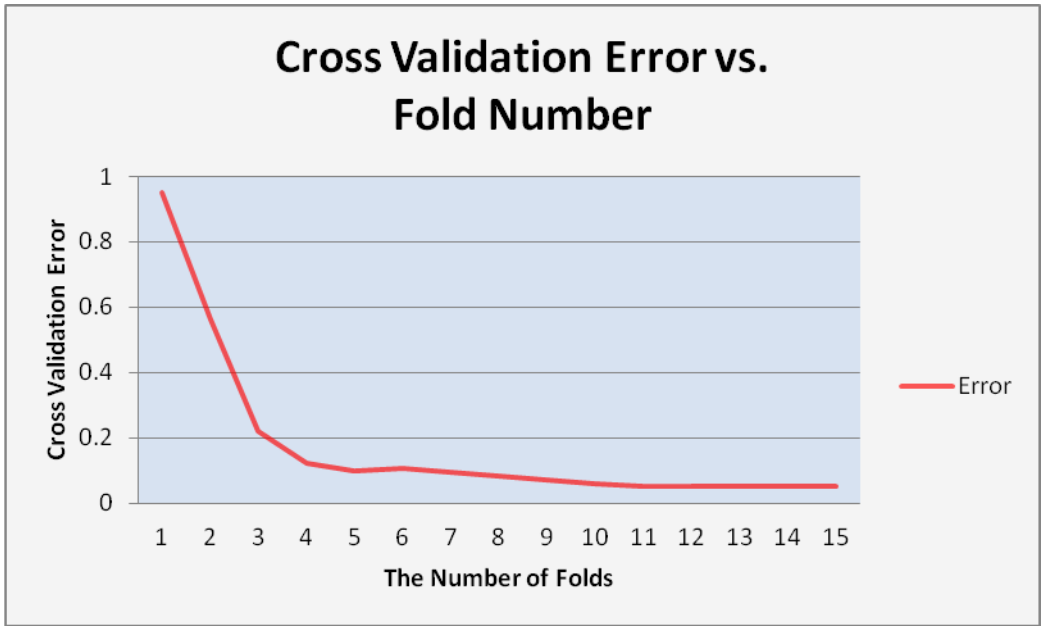


Figure 40. Cross Validation Error with Different Fold Number

We have explored different approaches to avoid overfitting. Early stopping is a very common practice in neural network training and often produces networks that generalize well. However, while often improving the generalization it does not do so in a mathematically well-defined way [85]. L2 norm is a popular approach to avoid overfitting and it produces better results in our experiment than L1 norm, which is known to generally give sparse feature vectors. Besides avoiding overfitting, cross validation is very useful for model selection, which has been shown in section 3.5.1. We use cross validation to determine the optimal number of hidden layers and number of neurons.

3.6 Conclusion

Boosting has demonstrated its success for binary classification problems, but little work has targeted traffic data analysis. In this research, boosting is introduced to travel

time prediction. Several different experiments are used to evaluate the feasibility of this algorithm. After comparing it with other approaches and implementing it under different iterations and data collection frequency, the applicability of boosting neural networks to travel time prediction is demonstrated.

Currently, this performance is based on experiments limited to travel time variation resulting from replicated simulated trials, potentially accounting for the relatively high prediction accuracy. Future work includes expanding the experimental design to include traffic demand variations typical of that found at similar time periods across different days and that found within a single day. The ultimate goal is to develop an algorithm sufficiently robust to capture non-recurring congestion and incidents.

CHAPTER 4

PARALLEL INTEREST MANAGEMENT AND FUTURE WORK

4.1 Summary and Future Direction

Motivated by two important requirements of Dynamic Data Driven Applications Systems (DDDAS) that incorporate on-line data into an executing application and utilize the analysis and prediction capabilities of simulations, this thesis has investigated research issues regarding interest management (data distribution) and prediction models.

In order to reduce communication overhead, we have proposed an interest management mechanism for mobile peer-to-peer systems that divides the entire space into cells and uses a bucket sort algorithm in each cell. The algorithm is applicable to dynamic sorting and matching of region modifications. If region modifications occur, the algorithm does not need to re-sort the list of projections and conduct the entire matching again. It can perform dynamic sorting and matching without re-processing all of the regions. This thesis also introduces the mobile landmarking design to implement this sort-based scheme in mobile peer-to-peer systems. The design is distributed and does not utilize fixed servers. Rather, each peer can become a mobile landmark node to take the server-like role to sort and match regions.

We have also explored the non-uniform distribution case. For the non-uniform distribution, random sampling is used to determine the boundaries of the bucket sort first. Then we use the bucket boundaries obtained in the random sampling phase to sort all the regions. During runtime, merging and splitting buckets are performed to maintain equi-depth buckets. Our new interest management mechanism is expected to have better computational efficiency for both static and dynamic matching. Experimental results

indicate that this approach yields better performance than several alternate interest management schemes.

In order to improve communication efficiency, the boosting approach is introduced as a method for travel time prediction in conjunction with neural network models to help capture characteristics of traffic and increase prediction accuracy. We have also examined the relationship between data collection frequency and travel time prediction accuracy, and proposed an approach to compute the lower bound of the collection frequency while maintaining high prediction accuracy, thereby introducing QoS as a factor in travel time prediction for the first time. Several different sets of experiments are used to evaluate the effectiveness of this model. After comparing it with other approaches and implementing it under different iterations and data collection frequencies, the results show that the boosting neural network model outperforms other predictors.

We have also examined the impact of different neural network settings on performance. We vary the number of hidden layers and the number of neurons to identify the appropriate FFNN structure and as expected, the two hidden layer FFNN gives better results when compared with the single layer FFNN. Different stopping criteria are applied to our model. Late stopping is used first to see when a minimum error on the training set is reached, and then early stopping is applied to avoid overfitting. Other approaches to avoid overfitting are also examined, incluing adding L1 norm, adding L2 norm and cross validation. Experimental results indicate that appropriate FFNN structure can provide better prediction accuracy before the boosting procedure. Results regarding regularization show that these approaches are efficient to make the model learn to generalize from trend rather than memorize training data.

There are several open issues with respect to interest management and travel time prediction models, some of which are listed as follows:

- **Mobile Server Mechanism Design**

In our current work, we have made several assumptions regarding mobile server mechanisms. At any time of the simulation, there is only one mobile landmark node (mobile server) for each cell. Before a mobile server leaves the cell, it will send messages to all the nodes in that cell requesting their distance to the landmark key. The mobile server will choose another node (which is now closest to the landmark key) to become the new mobile server and transmit the overlapping information in that cell to the new one. However, the current mechanism is based on the assumption that the mobile server is robust and communication with the mobile server is reliable. In real world scenarios, the mobile server itself and the communication with the mobile server may fail, thus multiple mobile servers should be allowed to exist in the same cell. In this case, many issues are interesting to explore, such as how to make multiple mobile servers have the same copy for one node (synchronization problem), how to distribute the workload to multiple mobile servers (load balancing problem), and what to do when a message is lost or when a mobile server fails (fault tolerance problem).

We also assumed the network is dense, meaning there is always a new mobile server available in each cell. If originally there is no mobile server in a cell, we need some mechanism to create mobile servers. In our current scheme, each mobile server sends beacons periodically within the cell in which this mobile server resides. Whenever a node hears a beacon, it stores the information of this mobile server. If there is no mobile server in a cell, when a node enters that cell, it will wait for some time for beacons (according to a threshold). If it does not receive any beacon within that period, it will become a new mobile server automatically and start to send beacons periodically within that cell. The exploration and analysis of this scenario is another direction of future work.

- **Parallel Interest Management**

Many existing interest management schemes focus on providing excellent message filtering accuracy, as well as seeking to reduce the computational overhead of matching. Computational performance is important, especially for some real-time applications where runtime performance is essential (such as massive multiplayer online games) and highly dynamic environments (such as intelligent transportation systems). For intelligent transportation systems, vehicles constantly move causing subscription and update regions to change quickly. Therefore, reducing the computational overhead for dynamic scenarios such as region modifications becomes an important problem.

Current interest management schemes are mainly designed for serial processing within each object. As the problem size grows, these algorithms do not scale well because serial processing may eventually become a bottleneck, especially for large-scale mobile systems. Therefore, implementing interest management schemes using parallel computation is a possible approach. A possible parallel interest management scheme will be presented in Section 4.3.

- **Including Traffic Demand Variations in the Prediction Model**

In our research, boosting is introduced for travel time prediction. Several different experiments are used to evaluate the feasibility of this algorithm. After comparison with other approaches and evaluating boosting under different iterations and data collection frequencies, the applicability of boosting neural networks to travel time prediction appears to have merit. Currently, this performance is based on experiments limited to travel time variation resulting from replicated simulated trials, potentially accounting for the relatively high prediction accuracy. Future directions include expanding the experimental design to include traffic demand variations typical of that found at similar time periods across different days and that found within a single day. The ultimate goal

is to develop an algorithm sufficiently robust to capture non-recurring congestion and incidents.

- **Exploring Different Basic Predictors**

Theoretically, the proposed boosting algorithm can use any basic learner. In our work, we utilize a feed-forward neural network (FFNN) and we also explored the impact of different neural network settings' on performance. One future direction is to use other machine learning methods as the basic learner and examine how they affect the final results. For instance, support vector machine (SVM) and linear regression are popular in machine learning research and they may be appropriate to use as the basic learner for boosting. Their prediction accuracy and training time compared to neural networks is another area of exploration.

4.2 Parallel Interest Management

4.2.1 Introduction

As discussed in chapter 2, interest management is essential for reducing communication overhead by filtering irrelevant messages in mobile distributed systems. Many existing interest management schemes focus on providing excellent message filtering accuracy, as well as seeking to reduce the computational overhead of matching. Computational performance is important, especially for some real-time applications where runtime performance is essential (such as massively multiplayer online games) and highly dynamic environments (such as intelligent transportation systems). For intelligent transportation systems, vehicles constantly move causing subscription and update regions to change quickly. Therefore, reducing the computational overhead for dynamic scenarios such as region modifications becomes an important problem.

We have proposed an interest management mechanism for mobile peer-to-peer systems using intelligent transportation systems as a motivating application. This mechanism is expected to have better computational efficiency for both static and dynamic matching. Experimental results indicate that this approach yields better performance than several alternate interest management schemes. However, for both our scheme and other existing interest management mechanisms, they are designed for serial processing within each object, such as serial processing in each mobile server in our mechanism. As the problem size grows, these algorithms do not scale because serial processing may eventually become a bottleneck, especially for large-scale mobile systems.

Several researchers have proposed parallel approaches for interest management mechanisms. In [65], the authors propose a parallel algorithm which can be run on shared-memory multiprocessors that improves the overall runtime efficiency of the filtering process. The process include two stages: the spatial decomposition stage and the interest matching stage. Spatial decomposition is used to divide the virtual space into a number of subdivisions and the interest matching process within one space subdivision is called a work unit (WU). WUs will be distributed across different processors for interest matching. For the interest matching stage, when a WU is being processed, a sort algorithm such as quick sort or insertion sort is used to determine the overlapping areas between subscription regions and update regions. While the second stage is similar with existing serial interest management algorithms, spatial decomposition is used to distribute the tasks among multiple processors, which is very important in parallel implementations.

In the spatial decomposition stage, a hash table is constructed so that each slot of the hash table represents a space subdivision. Then the subscription regions and update regions in that subdivision are inserted into the corresponding slot of the hash table. After this stage, each slot of the hash table represents a WU and all the WUs will be placed into a task queue. This facilitates load balancing. Each processor pops WUs from the task queue and performs the sorting and matching for the subscription regions and update

regions in the corresponding space subdivision. Figure 41 shows that the processors fetech from the head of the task queue.

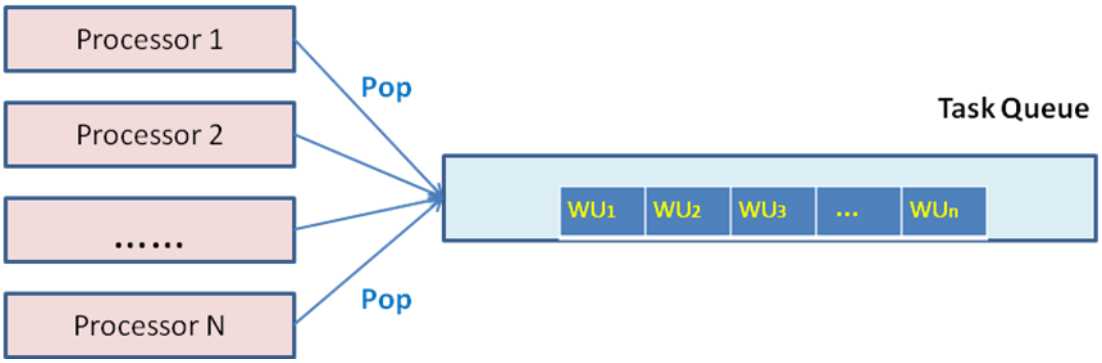


Figure 41. Task Queue for Load Balancing in Parallel Implementation

The algorithm in [66] is an extension of the above parallel algorithm that is suitable for distributed-memory systems. The proposed algorithm is designed to be generic and can run on different communication architectures, such as client-server with several nodes assisting the master server, multi-server and peer-to-peer architecture. No matter which architecture is used, since the algorithm runs on a cluster of computers that enables them to work simultaneously, the overall runtime efficiency of the sorting and matching process is enhanced. Experiments are based on MPI and show that this algorithm is more computationally scalable than one of the fastest serial algorithms.

An efficient parallel algorithm [67] for continuous interest management has also been proposed. The algorithm captures the missing events by using swept volumes to bound the trajectory of the regions and perform space-time overlap tests over each time interval. The workload of interest matching is distributed across multiple processors by this parallel algorithm. Since continuous interest management, swept volumes and space-time overlap tests are beyond the scope of the thesis, we will not discuss them further here.

Although current parallel interest management algorithms have much better computational efficiency than serial algorithms, they have drawbacks. First, current

parallel algorithms use CPU rather than GPU implementations. Second, current parallel algorithms use a task queue for load balancing, which does not scale. The blocking task queue performs poorly when faced with increasing numbers of processing units [68]. A non-blocking task queue yields better performance, but still scales poorly because the central queue remains a bottleneck when the number of processing units is increased. Third, the hash table is a redundant data structure. One work unit is placed into one slot of the hash table, and then placed into one position of the task queue from the hash table. All the relationships are one-to-one; one can store the information for one work unit into one position of the task queue directly.

To address these drawbacks, new approaches for parallel interest management may be considered. 1) Graphics processors rather than CPUs to speedup the implementation may improve scalability. 2) Task stealing is a popular load balancing mechanism, which performs very well and outperforms alternative approaches [68, 69]. 3) Double ended queues (deques) is an appropriate data structure. Deques are necessary for the task stealing scheme, and can be used in the space decomposition stage. After dividing the space into multiple divisions, the information corresponding to one processing unit can be put into one deque that will later be used in task stealing.

4.2.2 Parallel Interest Management Scheme

In this section, the design of the parallel interest matching scheme is presented. This scheme divides the interest matching process into initialization and runtime stages, and each stage includes multiple steps: initialized task distribution, task stealing and sorting and matching.

4.2.2.1 Initialized Task Distribution

As discussed in chapter 2, mobile landmarking is used in mobile peer-to-peer systems for sorting and matching. In Figure 42, the environment is partitioned into sixteen

cells and each landmark key is responsible for one cell. The node that is currently closest to a landmark key becomes a mobile landmark node. Then the second level space decomposition is used. Each cell is divided into M subcells and the tasks in M/N subcells will be distributed into one processing unit during the initialized task distribution stage (N is the number of processors). For example in Figure 42, M is equal to 16, which means each cell is divided into 16 subcells. Suppose N is equal to 4, then the tasks in 4 subcells will be processed in one processing unit. For instance, the tasks in subcells 1, 2, 3, 4 will be processed in processor 1, the tasks in subcells 5, 6, 7, 8 will be processed in processor 2 and so on.

Suppose N deques are used. The information related to subscription and update regions in M/N cells should be placed in one deque. For example, the projections on the x-axis and the y-axis of the subscription regions and update regions may be placed in deques, which is necessary for the following sorting and matching phase.

4.2.2.2 Task Stealing

Task stealing is an effective approach for load balancing [68, 69, 70]. Load balancing is very important for parallel implementation. If the workload is highly unbalanced, the advantage of parallel implementation cannot be exploited. In mobile peer-to-peer systems such as intelligent transportation systems, objects move continuously and dynamically and unbalanced workloads can occur with a high probability. For example, in intelligent transportation systems vehicles and their update regions may be grouped on the large arterial roads or ramp areas of highways while there are fewer vehicles and regions on small roads. Then the workload is unbalanced for different subdivisions in the space and load balancing schemes should be used among processors. In this scheme, each processing unit will process tasks in its local queue first. When the local queue is empty, it will try to steal a task from another processor's queue called the foreign queue. If a

processing unit creates a new task in the process, the new task is added to its own local queue.

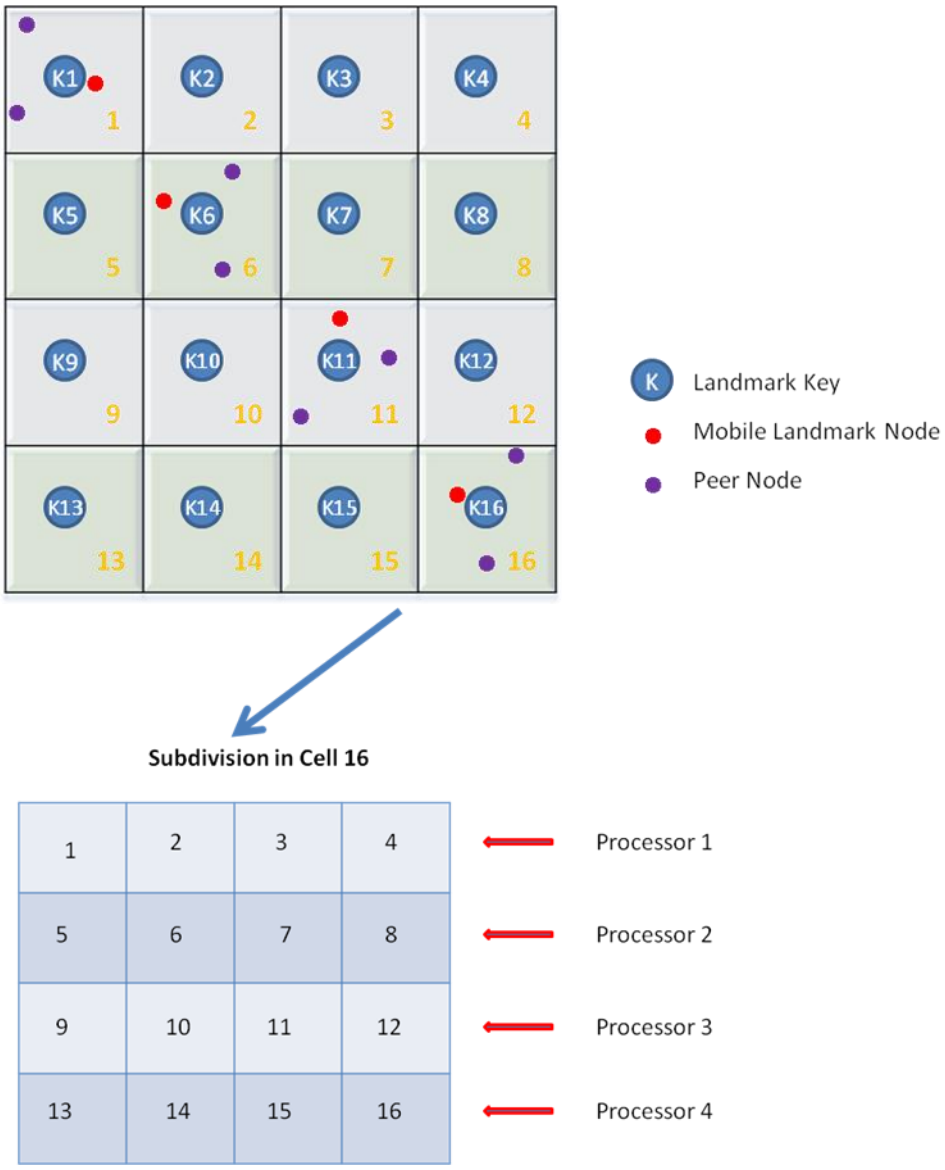


Figure 42. Initialized Task Distribution

The lock-free task stealing scheme [71] implemented with deques is an appropriate mechanism to use. The information about the objects in one subcell such as the projections of subscription regions and update regions will be pushed into the tail of its local deque. The information in the local deque will be poped from the tail and processed by the

corresponding processing unit. When the local deque is empty, the processing unit will try to steal from the head of another processor's deque.

To access the local deque, no lock or synchronization is needed since only the owner of this deque has the right to access it from the tail. However, with task stealing, multiple processing units may try to access the dequeue at the same time, necessitating synchronization [68]. But task stealing is only necessary when the local deque is empty and usually there are only few simultaneous accesses to a particular deque on systems with moderate parallelism [69], and hence the overall waiting time is small. However, if all the subscription regions and update regions reside in one subdivision, all the sorting and matching tasks will be located in one deque and performed in one single processor. The advantage of task stealing cannot be exploited in this scenario. This is the worst case, but also an extremely rare case.

The global round robin mechanism can be used for stealing the tasks. When processing unit *i* completes its own tasks and the local deque is empty, it looks at processing unit *i+1*. If the local deque of processing unit *i+1* is also empty, processing unit *i* will check processing unit *i+2*. The process is repeated until a deque with uncompleted tasks is found. Then process unit *i* will steal tasks from this foreign deque. The algorithm for task processing and task stealing is shown in Figure 43.

```
struct Task { Object Type, Attributes};

// 1. Initialization Phase
for (each subdivision i of the input data)
    correponding deque j.init(task.Object Type,
task.Attributes);

// 2. Working Phase
each processing unit j:
    loop
    {
        Task T deque j.get();
        if (!T)
      {
            for (k = processing unit j+1 to N-1)
        {
                Task T_new deque k.get();
                If(T_new)
                {
                    T_new.execute();
                  T_new.free();
                }
            }
            exit;
        }
        T.execute();
        T.free();
    }
```

Figure 43. Task Processing and Task Stealing

4.2.2.3 Sorting and Matching

During the sorting and matching phase, our interest management mechanism presented earlier can be used. A bucket sort-based algorithm is used for each subcell to compute the exact intersection between update and subscription regions. Specifically, our sort-based algorithm projects the regions on each dimension and uses bucket sort to sort the projections in order to find overlaps.

4.2.2.4 Runtime Stage

During the runtime stage, there may be dynamic modifications of subscription regions and update regions. Our previous scheme has the advantage of completing the dynamic sorting and matching without processing all of the regions, but unbalanced loads can still arise. For example, the objects in the corresponding subcells of processing unit *i* do not have region modifications while processing unit *i+1* is very busy with multiple region modifications, motivating the need for load balancing. One advantage of the task stealing mechanism is that it supports dynamic load balancing, which should be used in this dynamic scenario of region modifications. For dynamic load balancing, new tasks such as resorting the modified regions will be inserted to local deques and task stealing mechanism can process both initialized tasks and new tasks. For example, one deque is empty and the processing unit begins to steal tasks from another processing unit. At this time, new tasks are inserted into this deque and this processing unit will stop stealing tasks and give priority to its local tasks.

4.2.3 Proposed Implementation and Experiments

The parallel interest management scheme can be implemented using CUDA, which can compile C code into binary that can be executed on CUDA-enabled graphics processors. The NVIDIA 8600GTS and newer graphics processors support atomic operations such as CAS (Compare-And-Swap) and FAA (Fetch-And-Add) which can be used to implement efficient parallel data structures for the load balancing scheme [68].

The proposed scheme should be evaluated with several alternative mechanisms, such as 1) Parallel interest management with the blocking task queue load balancing approach. 2) Parallel interest management with the non-blocking task queue method. [65]. 3) Serial interest management scheme in our previous work [72]. The execution time and speed using each of the parallel methods should be recorded and compared while varying

the number of threads per block and blocks per grid. How the value of speedup changes with different numbers of threads and blocks can indicate the performance and scalability of each parallel mechanism.

The parallel scheme with blocking task queue is expected to have poor performance when the number of processing units increases because of spinning on the lock variable. These repeated attempts to acquire the lock cause the bus to be locked for large amounts of times [68]. The non-blocking queue method is expected to perform much better than the blocking task queue approach but the scalability will probably still be an issue due to the inherent central queue-based method. We expect that the proposed parallel scheme with task stealing has the best performance and scales well, since task stealing performs well for highly unbalanced workload scenarios and vehicle distribution as well as region distribution in intelligent transportation systems that have a high probability of being unbalanced.

4.3 Conclusion

This thesis has investigated two important research issues of Dynamic Data Driven Applications Systems (DDDAS): interest management (data distribution) and prediction models. This thesis has proposed an interest management mechanism for mobile peer-to-peer systems. A mobile landmarking design is also introduced to implement this sort-based scheme in mobile peer-to-peer systems. Experimental results indicate that this approach yields better performance than several alternate interest management schemes. This thesis has also presented the boosting approach as a method for travel time prediction in conjunction with neural network models to increase prediction accuracy. An efficient approach is also introduced to compute the lower bound of the data collection frequency while maintaining high prediction accuracy. The evaluation results show that the boosting neural network model outperforms other predictors.

There are several open issues which need to be explored with respect to interest management and travel time prediction models, such as mobile server mechanism design and parallel interest management. Including traffic demand variations in the prediction model and examining the performance of different basic learners are also interesting directions of future work.

REFERENCES

[1] Fujimoto, R.M., "Parallel and Distributed Simulation Systems," 2000: Wiley Interscience.

[2] Darema, F., "Dynamic data driven applications systems: A new paradigm for application simulations and measurements," Computational Science - ICCS 2004: 4th International Conference, 2004.

[3] https://www.dmso.mil/public/

[4] http://en.wikipedia.org/wiki/Peer-to-peer

[5] http://en.wikipedia.org/wiki/Distributed_hash_table

[6] Kuhl, F., Weatherly, R., Dahmann, J., "Creating Computer Simulation Systems," Prentice Hall, 2000.

[7] Stephen Ezell, "Explaining International IT Application Leadership: Intelligent Transportation Systems," Information Technology and Innovation Foundation, 2010.

[8] "Distributed Application Architecture," Sun Microsystem. June 2009.

[9] Ayani, R., F. Moradi, and G. Tan, "Optimizing cell-size in grid-based ddm," in PADS '00: Proceedings of the fourteenth workshop on Parallel and distributed simulation. Washington, DC, USA: IEEE Computer Society, 2000, pp. 93–100.

[10] Boukerche, A., N. J. McGraw, C. Dzermajko, and K. Lu, "Grid-Filtered Region-Based Data Distribution Management in Large-Scale Distributed Simulation Systems," in Annual Simulation Symposium, 2005, pp. 259–266.

[11] Daniel, S. R., and D. J. V. Hook, "Evaluation of grid-based relevance filtering for multicast group assignment," in Proceedings of the Distributed Interactive Simulation, 1996, pp. 739–747.

[12] Halevy, A. Y., Z. G. Ives, P. Mork, and I. Tatarinov, "Piazza: data management infrastructure for semantic web applications," in Internationl World Wide Web Conference, 2003, pp. 556–567.

[13] Hook, D. V., and J. Calvin, "Data distribution management in rti 1.3," in Proceedings of the 1998 Spring Simulation Interoperability Workshop, 1998.

[14] Hu, S. Y., J. F. Chen, and T. H. Chen, "Von: A scalable peer-to-peer network for virtual environments," IEEE Network, vol. 20, no. 4, pp. 22–31, 2006.

[15] Knutsson, B., H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in Proceedings of IEEE INFOCOM, 2004, pp. 96–107.

[16] Kubiatowicz, J., D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H.Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in ASPLOS, 2000, pp. 190–201.

[17] Liu, E. S,, and G. K. Theodoropoulos, "An Approach for Parallel Interest Matching in Distributed Virtual Environments," in Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009), Singapore, October 2009.

[18] Macedonia, M. R., M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups," IEEE Comput. Graph. Appl., vol. 15, no. 5, pp. 38–45, 1995.

[19] Pan, K., S. J. Turner, W. Cai, and Z. Li, "An Efficient Sort- Based DDM Matching Algorithm for HLA Applications with a Large Spatial Environment," in Proceedings of PADS '07, Washington, DC, USA, 2007, pp. 70–82.

[20] Pan, K., W. Cai, X. Tang, S. Zhou, and S. Turner, "A hybrid interest management mechanism for peer-to-peer networked virtual environments," in IEEE International Symposium on Parallel and Distributed Processing (IPDPS), 2010, pp. 1 – 12.

[21] Raczy, C., G. Tan, and J. Yu, "A sort-based ddm matching algorithm for HLA," ACM Trans. Model. Comput.Simul., vol. 15, no. 1, pp. 14–38, 2005.

[22] Ratnasamy, S., M. Handley, R. Karp, and S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," In IEEE Infocom '2002.

[23] Schroth, C., F. Dotzer, T. Kosch, B. Ostermaier, and M. Strassberger, "Simulating the traffic effects of vehicle-to-vehicle messaging systems," in Proc. of the 5th International Conference on ITS Telecommunications, Brest, France, 2005.

[24] Tan, G. S. H., Y. Zhang, and R. Ayani, "A Hybrid Approach to Data Distribution Management," in 4th International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2000), 2000, pp. 55–61.

[25] Wood, D. D.,"Implementation of DDM in the MAK High Performance RTI," in Proceedings of the 2002 Spring Simulation Interoperability Workshop, 2002.

[26] Yu, A. P., and S. T. Vuong, "Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games," in NOSSDAV '05: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video. New York, NY, USA: ACM, 2005, pp. 99–104.

[27] Juuso Aleksi Lehtinen, "Design and Implementation of Mobile Peer-to-Peer Application," Master's Thesis, Helsinki University of Technology, 2006.

[28] Michopoulos, Tsompanopoulou, Houstis, Joshi, "Agent-based Simulation of Data-Driven Fire Propagation Dynamics," Proceedings ICCS 2004.

[29] Douglas, Shannon, Efendiev, Ewing, Ginting, Lazarov, Cole, Jones, Johnson, Simpson, "A Note on Data-Driven Contaminant Simulation," Proceedings ICCS 2004.

[30] Sandu, Liao, Carmichael, Henze, Seinfeld, Chai, Daescu, "Computaional Aspects of Data Assimilation for Aerosol Dynamics," Proceedings ICCS 2004.

[31] Plale, B., D. Gannon, and D. Reed, "Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD," In International Conference on Computational Science. 2005.

[32] Hunter, M., H. K. Kim, W. Suh, R. Fujimoto, J. Sirichoke, and M. Palekar,"Ad Hoc Distributed Dynamic Data-driven Simulations of Surface Transportation Systems," Simulation 85:243–25, 2009.

[33] Hunter, M., J. Sirichoke, R. Fujimoto, and Y.-L. Huang, "Embedded Ad Hoc Distributed Simulation for Transportation System Monitoring and Control," In Proceedings of the 2009 INFORMS Simulation Society Research Workshop, edited by L. H. Lee, M. E. Kuhl, J. F. Fowler, and S. Robinson, 13–17. Hanover, MD: INFORMS, 2009.

[34] Low, M. Y. H. , K. W. Lye, P. Lendermann, S. J. Turner, R. T. W. Chim, and S. H. Leo, "An Agent-based Approach for Managing Symbiotic Simulation of Semiconductor Assembly and Test Operation," In Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems, 2005.

[35] Metaxas, Venkataraman, Vogler, "Image-Based Stress Recognition Using a Model-Based Dynamic Face Tracking System," Proceedings ICCS 2004.

[36] "Distributed Simulation Guide," Australian Defence Simulation Office, Department of Defense, 2004.

[37] J.W.C. van Lint, S.P. Hoogendoorn and H.J. van Zuylen, "Accurate travel time prediction with State-Space Neural Networks under missing data," Transportation Research Part C: Emerging Technologies, 2005.

[38] S. Lee and D.B. Fambro, "Application of subset autoregressive integrated moving average model for short-term freeway traffic volume forecasting," Transportation Research Record, pp. 179-188, 1999.

[39] M.Chen and I.J.Chien, "Dynamic freeway travel time prediction using probe vehicle data: Link-based vs. path-based," Transportation Research Record, National Research Council, Washiongton, D.C., 2001.

[40] X. Zhang and J. Rice, "Short-term travel time prediction using a timevarying coefficient linear model," Transportation Research: Part C, 2003.

[41] S. Clark, "Traffic prediction using multivariate nonparametric regression," Journal of Transportation Engineering, vol. 129, 2003.

[42] van Hinsbergen, C.P.I. and J.W.C. van Lint, "Bayesian combination of travel time prediction models," In 87th Annual Meeting of the Transportation Research Board, Washington DC, USA, 2008.

[43] J.W.C. van Lint, J. W. C., S. P. Hoogendoorn and H.J.van Zuylen. "Robust Freeway Travel time Prediction with State-Space Neural Networks," EWGT Mini Euro Conference, Polytechnic of Bari, 2003.

[44] C.H.Wu, J.M.Ho and D. T. Lee, "Travel-time prediction with support vector regression," IEEE Transactions on Intelligent Transporation Systems, 2004.

[45] R.E. Schapire, "The Boosting Approach to Machine Learning: an Overview," MSRI on Nonliear Estimation and Classification, 2002.

[46] R.Bone, M. Assaad, and M. Crucianu, "Boosting Recurrent Neural Networks for Time Series Prediction," Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms 2003.

[47] H. Drucker, "Improving Regressors using Boosting Teckniques," Proc. of the Fourteenth International Conference on Machine Learning, 1997.

[48] Y. Sun, "Cost-Sensitive Boosting for Classification of Imbalanced Data," Ph.D. thesis, Waterloo, Ontario, Canada, 2007.

[49] A. Khotanzad, and N. Sadek, "Multi-Scale High-Speed Network Traffic Prediction Using Combination of Neural Network," Proceedings of the International Joint Conference on Neural Networks, 2003.

[50] B. Kegl, "Robust Regression by Boosting the Median," Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, 2003.

[51] Y. Freund, "Boosting a weak learning algorithm by majority," In 3rd annual workshop on Computational Learning Theory. 1990.

[52] P. Buhlmann, and B. Yu, "Boosting with the L2-Loss:regression and classification," Research Report, June. 2002.

[53] Ishak, S. and Al-Deek, H., "Performance evaluation of short-term time-series traffic prediction model," Journal of Transportation Engineering, 2002, Vol. 128, No. 6, 490-498.

[54] Bo Xu, Ouri Wolfson, "Time-Series Prediction with Applications to Traffic and Moving Objects Databases," MobiDE'03.

[55] Sherif Ishak and Haitham Al-Deek, "Performance Evaluation of Short-Term Time-Series Traffic Prediction Model," Journal of Transportation Engineering, 2002.

[56] Oda, T., "An Algorithm for Prediction of Travel Time Using Vehicle Sensor Data," Proceedings of the IEEE 3rd International Conference on Road Traffic Control, London, 1990, pp.40-44.

[57] Al-Deek, H., M. D'Angelo and M. Wang, "Travel Time Prediction with Non-Linear Time Series," Proceedings of the ASCE 1998 5th International Conference on Applications of Advanced Technologies in Transportation, Newport Beach, CA, 1998, pp.317-324.

[58] D. Montgomery, L. Johnson, and J. Gardiner, "Forecasting and Time Series Analysis.," McGraw-Hill, 1990.

[59] Tong, H., "Nonlinear time series: a dynamic approach," Oxford University Press, New York, 1993.

[60] Billi, M., Ambrosino, G. and Boero, M., "Artificial Intelligence Applications to Traffic Engineering," CRC Press, 1994.

[61] Wei, C.-H., Lin, S.-C. and Li, Y, "Empirical Validation of Freeway Bus Travel Time Forecasting," Transportation Planning Journal, 2003, Vol. 32, 651-679.

[62] Jiang, G. and Zhang, R., "Travel Time Prediction for Urban Arterial Road: A Case on China," Proceedings of Intelligent Transport System, IEEE, 2001, 255-260.

[63] Jiang, G. and Zhang, R., "Travel Time Prediction for Urban Arterial Road," Proceedings of Intelligent Transport System, IEEE, 2003, 1459-1462, 12-15.

[64] Kisgyorgy, L. and Rilett, L.R., "Travel Time Prediction by Advanced Neural Network," Periodica Polytechnica Civil Engineering, 2002, Vol. 46, No. 1, 15-32.

[65] E. S. Liu and G. K. Theodoropoulos, "An Approach for Parallel Interest Matching in Distributed Virtual Environments," in Proceedings of the 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2009), Singapore, October 2009.

[66] E. S. Liu and G. K. Theodoropoulos, "A Parallel Interest Matching Algorithm for Distributed-Memory Systems," in Proceedings of the 15th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2011.

[67] E. S. Liu and G. K. Theodoropoulos, "A Fast Parallel Matching Algorithm for Continuous Interest Management," in Proceedings of the Winter Simulation Conference, 2010.

[68] Daniel Cedermany and Philippas Tsigas, "On Dynamic Load Balancing on Graphical Processors," In Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, 2008.

[69] Matthias Korch and Thomas Rauber, "A Comparison of Task Pools for Dynamic Load Balancing of irregular algorithms," Concurrency and Computation: Practice & Experience archive Volume 16 Issue 1, Page 1-47, December 2003.

[70] Stanley Tzengy, Anjul Patney and John D. Owens, "Task management for irregular-parallel workloads on the GPU," in Proceedings of the Conference on High Performance Graphics, 2010.

[71] Arora N. S., Blumofe R. D., Plaxton C. G., "Thread Scheduling for Multiprogrammed Multiprocessors," in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, 1998.

[72] Ying Li, Richard Fujimoto, Michael Hunter, Wonho Suh, "An Interest Management Scheme for Mobile Peer-to-Peer Systems," in Proceedings of Winter Simulation Conference, 2011.

[73] M. Muralikrishna and David J Dewitt, "Equi-depth histograms for estimating selectivity factors for multi- dimensional queries," In Proceedings of ACM SIGMOD Conference, 1988.

[74] W. D. Frazer and A. C. McKellar, "Samplesort: A Sampling Approach to Minimal Storage Tree Sorting," Journal of the Association for Computing Machinery, July 1970.

[75] Hamid Mousavi and Carlo Zaniolo, "Fast and Accurate Computation of Equi-Depth Histograms over Data Streams," EDBT 2011, March 22–24, 2011, Sweden.

[76] A. I. McLeod and D. R. Bellhouse, "A Convenient Algorithm for Drawing a Simple Random Sample," Applied Statistics, 1983, Volumn 32, Issue 2, 182-184.

[77] Phillip B. Gibbons, Yossi Matias and Viswanath Poosala, "Fast Incremental Maintenance of Approximate Histograms," Proceedings of the 23rd VLDB Conference, 1997, Greece.

[78] Anderson, J., M. Bell, T. Sayers, F. Busch, and G. Heymann, "The Short-Term Prediction of Link Travel Time in Signal Controlled Road Networks," Proceedings of the IFAC/IFORS 7th Symposium on Transportation Systems: Theory and Application of Advanced Technology, Tianjin, China, 1994, pp.621- 626.

[79] D'Angelo, M., H. Al-Deek, and M. Wang, "Travel Time Prediction for Freeway Corridors," 78th TRB Annual Meeting, Washington, DC, 1999.

[80] A. Sen, A., N. Liu, P. Thakuriah, and J.Li, "Short-Term Forecasting of Link Travel Times: A Preliminary Proposal," ADANCE Working Paper Series #7, 1991.

[81] Dongjoo Park, Laurence R. Rilett, and Gunhee Han, "Spectral Basis Neural Networks for Real-Time Travel Time Forecasting," Journal of Transportation Engineering, Vol. 125, No. 6, 1999.

[82] Karl Nygren, "Stock Prediction – A Neural Network Approach," Master Thesis, Royal Institute of Technology, KTH, 2004.

[83] http://en.wikipedia.org/wiki/Overfitting

[84] Patrick Sterlin, "Overfitting Prevention with Cross-Validation," Supervised Machine Learning Report, 2007.

[83] http://en.wikipedia.org/wiki/Early_stopping

VITA

YING LI

YING LI was born in Tianjin, China. She attended public schools in her hometown and received a B.S. in Computer Science from Nankai University in 2007. She came to Georgia Tech to pursue a doctorate in Computer Science after obtaining the bachelor's degree. Ying also received her M.S. in Computer Science from Georgia Tech in August 2012. She did two internships during her PhD study. When she is not working on her research, Ying enjoys listening to music, hiking and cooking delicious food.