FHWA/IN/JTRP-2001/7

Final Report

VALIDATION, CALIBRATION, AND EVALUATION OF ITS TECHNOLOGIES ON THE BORMAN CORRIDOR

Srinivas Peeta
Raghubushan Pasupathy
Pengcheng Zhang

January 2002

Final Report

FHWA/IN/JTRP-2001/7

Validation, Calibration, and Evaluation of ITS Technologies on the Borman Corridor

by

Srinivas Peeta
Principal Investigator
Associate Professor of Civil Engineering

and

Pengcheng Zhang
and
Raghubushan Pasupathy
Graduate Research Assistants

School of Civil Engineering
Purdue University

Joint Transportation Research Program
Project No. C-36-75M
File No. 8-9-13
SPR 2208

Prepared in cooperation with the
Indiana Department of Transportation and the
U.S. Department of Transportation
Federal Highway Administration

Purdue University
West Lafayette, Indiana 47907
January 2002

ACKNOWLEDGEMENTS

The authors acknowledge the assistance and feedback of the members of the study advisory committee: David Boruff, Shou Li, Scott Newbolds, Mark Newland, and Wes Shaw from the Indiana Department of Transportation, and Clemenc Ligocki of the Federal Highway Administration. The authors would also like to thank Troy Boyd of the Indiana Department of Transportation for his assistance in providing information and feedback on the Borman advanced traffic management system. The authors further acknowledge the contributions of graduate research assistants Ioannis Anastassapoulos and Jeong Whon Yu to the research described in this report.

This project was funded by the Joint Transportation Research Program of Purdue University in cooperation with the Indiana Department of Transportation and the Federal Highway Administration. We appreciate their support and assistance.

INDOT Research

TECHNICAL *Summary*

Technology Transfer and Project Implementation Information

TRB Subject Code 12-5 Transportation Systems and Technology *January 2002*
Publication No.: FHWA/IN/JTRP-2001/07, SPR-2399 *Final Report*

VALIDATION, CALIBRATION, AND EVALUATION OF ITS TECHNOLOGIES ON THE BORMAN CORRIDOR

Introduction

The spawning of the Internet and the communication revolution, coupled with highly economical computing commodity hardware costs, are motivating new paradigms for the real-time operation and control of large-scale traffic systems equipped with advanced information systems and sensor technologies. Non-availability of reliable on-line data and the prohibitive costs of high performance computers to process such data have previously been the primary barriers in these endeavors. Recently, falling computer hardware costs, the exponential growths in their performance capabilities, sophisticated sensor systems, and the ability to transmit data through the public domain quickly and reliably, have been synergistic in enabling the efficient deployment of real-time route guidance in large-scale traffic systems.

This study develops an Internet-based on-line architecture to exert control in large-scale traffic systems equipped with advanced sensor systems and information dissemination media. The aim is to develop an automated on-line system that disseminates messages to network users on the optimal paths and/or provides route guidance while satisfying accuracy and computational efficiency requirements. It explicitly accounts for the calibration and consistency-checking needs of the models being used within the architecture. The architecture incorporates fault tolerance methods for errors encountered at the architecture level and due to the malfunctioning of field sensors. Since cost is an important factor for large-scale deployment, an Internet-based remote control architecture is proposed to enable deployment and evaluation. The architecture is remote in that it can be used to deploy and evaluate alternative solution strategies in the offline/online modes from remote site locations (such as the Borman or Indianapolis traffic control centers). In addition, a remote architecture ensures that the associated models can be located at one central server and accessed from any INDOT site.

The study also proposes to use the Beowulf Cluster as an economical, flexible, and customizable computing paradigm to generate supercomputing capabilities for the real-time deployment of the proposed on-line control architecture. It serves as the enabling environment to execute and coordinate the activities of the various modules responsible for real-time network route guidance, data transmission, calibration and fault tolerance. In the context of large-scale traffic systems, a Beowulf Cluster can be configured in centralized as well as decentralized on-line control architectures with equal ease. Thereby, it enables individual traffic operators with smaller operational scope (such as local traffic agencies) to install mini Beowulf Clusters at their locations or allows several of them to operate remotely using a centrally located large-scale Cluster. For a large transportation agency, the use of a centrally located Cluster can significantly aid operational efficiency and cost reduction by obviating the need for hardware, software, space requirements, and maintenance at each individual location.

Findings

This research has the following findings that meet the research objectives.

1. An Internet-based remote traffic control architecture is economical and efficient, obviates redundancies in hardware and maintenance, and can be automated by incorporating fault tolerant systems.
2. The Beowulf Cluster computing paradigm serves as a viable alternative to expensive specialized supercomputing architectures to address the computational needs of real-time traffic operations and control.
3. Beowulf Clusters can be configured with equal ease for centralized and decentralized on-line control architectures. This enables small as well as large traffic agencies to implement a new generation of robust, but computationally burdensome, methodologies for traffic operations based on the application of advanced technologies.
4. The Fourier transform-based fault tolerant framework can detect data faults due to malfunctioning detectors and predict the likely actual data for the seamless operation of an on-line traffic control architecture. It can also detect incidents.

The off-line benchmarking tests to analyze data communication and parallel software codes suggest that the proposed Cluster computing architecture is highly efficient, and can enable real-time deployment of the associated traffic control strategies.

Implementation

The proposed on-line traffic control architecture can be implemented on the Borman Expressway or Indianapolis ATMS corridors for real-time route guidance operations after exploring various issues for enabling real-time communication links.

Contact

For more information:

Prof. Srinivas Peeta
Principal Investigator
School of Civil Engineering
Purdue University
West Lafayette IN 47907
Phone: (765) 494-2209
Fax: (765) 496-1105

Indiana Department of Transportation
Division of Research
1205 Montgomery Street
P.O. Box 2279
West Lafayette, IN 47906
Phone: (765) 463-1521
Fax: (765) 497-1665

Purdue University
Joint Transportation Research Program
School of Civil Engineering
West Lafayette, IN 47907-1284
Phone: (765) 494-9310
Fax: (765) 496-1105

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/IN/JTRP-2001/7		

4. Title and Subtitle	5. Report Date
Validation, Calibration, and Evaluation of ITS Technologies on the Borman Corridor	January 2002
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Srinivas Peeta, Raghubushan Pasupathy, Pengcheng Zhang	FHWA/IN/JTRP-2001/7

9. Performing Organization Name and Address	10. Work Unit No.
Joint Transportation Research Program 1284 Civil Engineering Building Purdue University West Lafayette IN 47907-1284	
	11. Contract or Grant No. SPR-2399

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
Indiana Department of Transportation State Office Building 100 North Senate Avenue Indianapolis, IN 46204	Final Report
	14. Sponsoring Agency Code

15. Supplementary Notes

Prepared in cooperation with the Indiana Department of Transportation and Federal Highway Administration.

16. Abstract

This study develops an Internet-based remote on-line traffic control architecture for route guidance in large-scale traffic systems equipped with advanced information systems and sensors. It also proposes to use the Beowulf Cluster paradigm as an economical, flexible, and customizable computing architecture to generate supercomputing capabilities within the control architecture. The Beowulf Cluster provides the enabling environment to execute and coordinate the activities of the various modules that address real-time network route guidance, data transmission, calibration and fault tolerance. In the context of large-scale traffic systems, a Beowulf Cluster can be configured in centralized as well as decentralized on-line control architectures with equal ease. Thereby, it enables individual traffic operators with smaller operational scope (such as local traffic agencies) to install mini Beowulf Clusters at their locations or allows several of them to operate remotely using a centrally located large-scale Cluster. To enable automation and the seamless operation of the on-line architecture, a Fourier transform-based fault tolerant framework is introduced that can detect data faults due to malfunctioning detectors and predict the likely actual data on-line.

17. Key Words	18. Distribution Statement
Internet-based On-line Traffic Control Architectures, Beowulf Clusters, Real-time Route Guidance Deployment, Fault Tolerance, Automatic Detection of Sensor Malfunctions, Fourier-based Short-term Traffic Predictions, Real-time Traffic Data Communication and Storage	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	100	

Form DOT F 1700.7 (8-69)

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1. INTRODUCTION

1.1 Background and Problem Statement

Several intelligent transportation (ITS) technologies are being employed or are currently under deployment along a number of freeway corridors in Indiana. They include advanced traffic management systems (ATMS) and advanced traveler information systems (ATIS) on I-80/94 (Borman expressway) in northern Indiana, in the greater Indianapolis area, and along the Indiana/Kentucky corridor. In addition, real-time incident management operators called the Hoosier Helpers are currently operational on the Borman expressway and in northeastern Indianapolis. Incident detection and integrated incident detection-response technologies are nearing full deployment on the Borman expressway. A continuous capability to validate, calibrate, and evaluate the deployed technologies and related strategies is necessary to significantly enhance the effectiveness of the ITS program in Indiana, and provide guidelines for future deployments. Till recently, the sparseness of actual field data has hampered the ability to predict potential deployment impacts with a high level of confidence. This tends to be a general issue nationwide. Simulation and/or limited data collection methods have been used to partly alleviate this crucial problem. However, the implementation of advanced technologies in recent years under the auspices of ITS provides an ability to generate huge amounts of detailed and more accurate time-dependent data.

A related problem of critical importance to the successful deployment of ITS technologies is the ability to quickly process huge amounts of these real-time data to generate on-line solution strategies (for example, detour information provision, route

guidance, dynamic signal systems control) to manage large congested traffic networks. Off-line solutions are either restrictive or involve massive computational efforts in considering multiple plausible scenarios. They may serve as initial solutions on-line. However, the on-line solution is characterized by the need to be responsive to unfolding on-line conditions, thereby requiring the computation of updated solutions in sub-real time. It inherits a host of other issues arising from the on-line nature. Consistency issues arise because models are abstractions of real world phenomena and suffer from inaccuracies introduced by simplifying assumptions and other factors. The models need to be updated on-line to ensure realistic representation of the actual traffic conditions. Another aspect is the need for fault tolerant mechanisms that account for possible failure modes in such on-line systems. The associated procedures aim at providing fallback strategies when essential system components fail. This is especially important for automated real-time systems.

The recent spawning of the Internet, and the communication revolution, are motivating new paradigms for the real-time operation of large-scale traffic systems equipped with advanced technologies. Non-availability of reliable on-line data and the prohibitive costs of high performance computers to process such data have previously been the chief impediments in these endeavors. More recently, falling computer hardware costs, the exponential growths in their performance capabilities, sophisticated sensor systems, and the ability to transmit data through the public domain quickly and reliably, have been synergistic in enabling the efficient deployment of large-scale traffic system operations such as real-time route guidance.

1.2 Study Objectives

The primary objective of this study is to develop an internet-based on-line architecture to exert control in large-scale traffic systems equipped with advanced sensor systems and information dissemination media. The aim is to develop an automated on-line system that disseminates messages to network users on the optimal paths and/or provides route guidance while addressing the on-line issues discussed earlier. Since economic efficiency is an important requirement for large-scale deployment of real-time traffic systems, the study seeks to develop an internet-based remote control architecture to aid and enable ITS deployment and evaluation. The architecture is remote in that it can be used to deploy and evaluate alternative solution strategies in the offline/online modes from remote site locations (such as the Borman expressway or the Indianapolis traffic control centers). In addition, a remote architecture ensures that the associated models can be located at one central server and accessed from any site in the state.

This study also proposes to use the Beowulf Cluster as an economical, flexible, and customizable computing architecture to generate supercomputing capabilities within the control architecture. The Beowulf Cluster as the computing paradigm in the on-line architecture will serve as the environment that will execute and coordinate the activities of the various modules responsible for real-time network route guidance, data transmission, calibration and fault tolerance.

1.3 Organization of the Report

The report consists of six chapters. Chapter 2 describes the on-line architecture for real-time traffic systems. Chapter 3 discusses the Beowulf Cluster computing paradigm and

describes in detail the hardware, software and data transmission mechanisms that were used in the design of a sixteen processor Beowulf cluster called Super Computing Cluster for On-line and Real-time Control of Highways using Information Technology (SCORCH-IT). Chapter 4 describes some of the fault tolerance issues in the context of on-line and real-time traffic systems. Some off-line tests that were performed to evaluate the functioning of the on-line architecture and the Beowulf computing paradigm are reported in Chapter 5. Chapter 6 provides some concluding remarks and recommendations.

2. ON-LINE ARCHITECTURE FOR REAL-TIME TRAFFIC SYSTEMS CONTROL

This chapter describes an internet-based on-line architecture to exert control in large-scale traffic systems equipped with advanced sensor systems and information dissemination media. The objective is to develop an automated on-line system that disseminates messages to network users on the optimal paths and/or provides route guidance while addressing the on-line issues discussed earlier. The chapter is organized into two parts: the first part describes the various components of the on-line architecture and the associated logic. The second part describes the computing paradigm that enables efficient on-line implementation of the on-line architecture.

2.1 On-line Architecture

The on-line architecture, shown in Figure 2.1, is described in terms of (i) the main components of the architecture, and (ii) the flow logic that integrates these components.

2.1.1 Traffic Control Center

The traffic control center is the traffic controller. The controller aims at enhancing the network performance through information dissemination strategies that provide routing information to network users. The traffic controller is a significant functional component of the architecture because the controller is the system operator. Thereby, the on-line network conditions are significantly influenced by the controller's actions.

2.1.2 Information Dissemination Strategies

They represent the strategies implemented by the traffic controller on the network. The controller can disseminate information through one or more of several available

media, e.g. in-vehicle navigation systems, dynamic message signs, radios etc. Also different messages may be used to target different groups of users based on their behavioral tendencies. These strategies represent the primary control mechanism for the traffic controller. However, other control mechanisms can be easily incorporated, as the architecture is not mechanism specific.

2.1.3 On-Line Traffic Network

Real-time data from various sensors on the traffic network is a critical component of the proposed on-line architecture. Hence, changing traffic conditions, data type, measurement mechanisms, transmission issues, and associated failure modes define the on-line nature of the traffic network.

2.1.4 Virtual System Simulation

This component runs in real-time and aims to faithfully replicate actual traffic conditions. It is an essential element vis-à-vis fault tolerance and is used to generate fallback control strategies under failure modes.

2.1.5 Calibration and Consistency Checking

This component aims at bridging the gap between actual data measurements of traffic conditions and the traffic conditions predicted by the models embedded in the control logic. It seeks to re-calibrate models on-line to enhance the prediction accuracy of evolving traffic conditions.

2.1.6 Control Models

They represent the collection of tools necessary to generate the solution strategies to be provided to the traffic controller. They use data from the traffic system, feedback from the consistency module, and historical trends to specify the on-line control

strategies. This is computationally the most intensive part of the architecture and is the focal point for the design of efficient computing paradigms.

2.1.7 Alternative Solution Strategies

The solution strategies represent the output of the control components and the primary input to the traffic controller. These strategies are transmitted to the traffic controller to begin a new cycle of operations.

2.1.8 The Flow Logic

The flow logic for the on-line architecture can be differentiated based on the normal and failure modes. The normal mode refers to the situation when all components of the on-line architecture are functioning normally. In this mode the traffic network conditions in response to the control strategies implemented by the control center are measured using the sensor system. Real-time traffic data obtained from detectors and data on the information dissemination strategies employed is sent via communication channels (such as the internet) to the computing and processing unit, which may be located at a different site. Simultaneously, it is also archived in the traffic control center's database for possible future use. The processing unit, typically a dedicated computer server, is the location where the major computational components such as the virtual system simulation, calibration and consistency checking modules, and the control models, are executed. Figure 2.1 illustrates the flow logic for the computing and processing unit. The output from the consistency module is used to update model parameters in the control models. The system state is updated using the modified parameters in the virtual simulator. The associated data is transmitted to the control models. The solution strategies generated using the control models are then transmitted to the traffic control

center using communication channels such as the internet to complete a typical flow cycle under normal circumstances. The failure mode is caused either due to the failure of the communication links, sensors, and/or the computing hardware. When the communication link (internet) that transmits data to the processing unit from the traffic control center fails, data is obtained directly from the traffic control center database through a dedicated communication channel (such as a dial-up connection). Another failure scenario is when a detector located on the field malfunctions, and is no longer able to transmit data to the traffic control center. In such a case, the control models use data from the virtual system simulator to determine solution strategies. The calibration and consistency checking modules are by-passed in the failure mode as real-world data is unavailable.

2.2 Computing Paradigm

Due to the significant computational intensity of the various components of the processing unit and the need for real-time or sub real-time solutions for the traffic control center, a computing environment is desirable that is capable of efficient and cost-effective high performance computing. Traditionally, this role has been the domain of prohibitively expensive supercomputers. The recent drastic reduction in computer hardware costs coupled with exponential increases in computing capabilities motivate the development of a Cluster computing architecture that is customized to the problem being addressed, and can approach supercomputing capabilities. We propose the use of a Beowulf Cluster that has several advantages in the operational context of on-line systems. A Beowulf Cluster is a distributed computing system consisting of several nodes

connected together through standard Ethernet adapters and switches. A node here refers to one physical machine containing one or more processors. The system includes a master node and several client nodes that receive instructions from the master. Characteristics that facilitate high performance computing include:

- Low communication overheads between the various nodes of the Cluster, made possible through fast ethernet switches.
- An architecture that is customized to the problem being addressed. For example, if the problem requires large input-output capabilities, the associated hardware can be selectively included.

Dedicated stand-alone architecture that circumvents the vagaries of network traffic thereby enhancing reliability and computational efficiency. The architecture of typical Beowulf Clusters, associated hardware and software configuration, and data storage and performance issues will be discussed in detail in Chapter 3. The various functions of the Beowulf Cluster performed over the typical time cycle of the computing system are illustrated in Figure 2.2. The four layers of rectangles represent specific functions of the Cluster. All distinct functions in a layer are executed sequentially. However, tasks of two different layers may be executed concurrently if they share the same timeline. The first layer is data related and includes data retrieval, formatting, and storage. Since the master node is the Cluster's only gateway to the outside world (for security reasons), it performs the function of continually receiving field traffic data. However, if the data retrieval is the computational bottleneck in the time cycle, additional nodes may be used to receive the data. The data received is formatted for use by the virtual system simulation and the control models. Here as well, the task may be assigned

to one or more nodes. In order to maximize efficiency in the Cluster, it is advisable not to pre-assign tasks to individual nodes. Instead, tasks can be distributed dynamically based on the current load on each node. The data storage task involves storing the data at different nodes in the Cluster based on future data retrieval needs. If the tasks of the various nodes are not fixed, an optimal allocation strategy for data storage may not exist. Under such a framework, a node with large disk space can be dedicated to data storage.

Load balancing is critical to the computational efficiency of the Cluster. Typically performed by the master node, it focuses on the mechanism to assign individual tasks of a process to various nodes so that the execution time is minimized. Hence, load balancing is the rule by which a process is decomposed and distributed in the Cluster. Apart from optimization at the Cluster level, tasks can be optimized at the individual node level. This is possible if a node contains more than one processor. The associated procedure, called multi-threading, uses the fact that multiple processors share the same memory pool resulting in fast communication and synchronization between these processors. Hence, large tasks that involve little global data can be executed on multi-processor nodes.

The other functions in this layer include visualization and output formatting and transmission. Visualization is the animation component of the application and is performed by a specific node using the output data. Simultaneously, the output data containing the dissemination strategies to be deployed on-line is encrypted and transmitted through the Internet back to the traffic control center.

Since the Beowulf Cluster is employed for on-line operations using the on-line architecture, fault tolerance is an important functionality. Fallback strategies are necessary to address node failures. A fault tolerance mechanism will warn the master

node about imminent node failure so that its tasks can be re-allocated. Some Clusters have multiple master nodes to address the possibility of master node failure. The fault tolerance rectangle in Figure 2.2 covers the entire cycle implying that this function is performed at all times.

The virtual system simulation is another function that is performed by the Cluster at all times. As discussed earlier, the virtual simulator provides data to the control models when either the detectors or the communication link to the Cluster fail.

2.3 Data Routing Architecture

One of the key issues in the on-line traffic control architecture is the transmission and acquisition of traffic-related data such as volume, density and speed as detected by the sensors installed in the roadway. These data will be collected by local Traffic Control Center (TCC), and be processed to generate real-time traffic control strategies, which aim to optimize network performance. A reliable, fast, secure, and cost efficient data routing architecture is key to the efficient functioning of the Internet-based on-line control architecture.

In earlier applications, remote data acquisition and communication to capture data in multiple locations have been enabled either by leaving standalone equipment and collecting the data periodically, or by using expensive wired or radio links. Until early 1990's, data acquisition and logging have required custom hardware and software solutions, which have been prohibitively expensive for large-scale implementations such as multi-pointed field traffic data collection. During the past decade, PC-based

instrumentation appeared on the scene, which have brought significant improvement on data acquisition.

2.3.1 Internet-based Data Routing for On-line Control Traffic Architecture

Data acquisition could reap the benefits of low cost and standardized user interfaces due to the spawning of Internet in recent years. The Internet has provided access between networks separated by large geographical distances. The use of Internet for data acquisition provides a number of advantages, a few of which include: 1) real-time access to data for more than one individual user; 2) access to a number of data loggers for each individual user; 3) relatively low cost; 4) abundant software resources and strong technical support; 5) mature technologies and communication architectures such as TCP/IP, Sockets, FTP, and distributed database; and 6) facility to change the operating characteristics of a data logger remotely.

Figure 2.3 shows an Internet-based data acquisition architecture. The data flow could be readily grouped into four phases. Phase 1 is termed “raw data acquisition”. In this phase, traffic related data are collected by field sensors such as video camera, infrared sensors, magnetic detectors, microwave detectors, or loop detectors. These data will be transmitted to a local traffic control agency in real-time for further process, distribution, or storage. Normally there are dedicated links between field sensors and the TCC, and the data could also be transmitted through wireless communication such as satellite or microwave. Phase 2 is called “computational data acquisition” and involves the transmission of data between TCC and a computing environment called the “computational unit”. This unit typically possesses powerful computational capabilities and is embedded with traffic optimization algorithms. Data will be transmitted from the

TCC to the computational unit and processed by relevant algorithms. Corresponding control strategies such as shortest path information, route guidance, and traffic signal setting update plans will be generated and sent to the TCC for traffic network control. The computational unit could potentially be located locally with the TCC, in which case the data can be carried through fast Ethernet thereby enhancing the communication speed and security; or located remotely, in which case the data could be transmitted through the Internet. The advantages of the latter case is that multiple users can share the same hardware, software, and personnel resources, which facilitate the update and maintenance of the system, and make full use of expensive resources. Phase 3 is called "real-time traffic information acquisition". The real-time traffic control strategies generated in computational unit will be applied into the network. Traffic-related information will be provided to drivers, freight carriers, police, and other end users or managers of transportation systems through VMS, in-vehicle devices, public media, Internet, or wireless communications facilities. The communication will be carried through satellite, microwave or dedicated lines. Phase 4 is called "off-line data acquisition". Historical traffic data stored in the TCC database can be accessed by traffic planners, insurance companies, local traffic engineers or any other qualified users in an off-line manner.

2.3.2 Data Acquisition System Design

Figure 2.4 shows a realization of data acquisition architecture. The hardware components of such an architecture include (i) On-site traffic condition detectors, (ii) Data acquisition domains, (iii) Data processing units, (iv) End-user facilities and (v) Data communication connections. These components are collectively responsible for the gathering and processing of real-time traffic-related data, generation of real-time traffic

control strategies, storage and dissemination of advisory and/or guidance information to end-users. An important issue in the data acquisition architecture is the remote Data Acquisition System (DAS) design, which will be discussed in detail in the following section.

2.3.3 Data Acquisition Architecture

At each end of server domains, a DAS is used to control the data acquisition process. Each DAS will have a two-way communication channel with the server. The acquired data is transmitted over this channel, which is then archived on the server. The server can also send commands over the channel to the individual DAS to change their operating characteristics. The archived data can be accessed by the clients, and at the same time instruct the server to change operating characteristics of any DAS. In a DAS, the data acquisition hardware is controlled by a host computer. Figure 2.5 shows the principal functional units of a DAS.

There are a wide variety of ways in which communication between the PC and DAS hardware can be accomplished. The most common interface is the type of cards that "plug in" directly into the computer bus within the computer case. The main advantage of such an interface is that it provides both a high bandwidth and the ability to apply the host's computational resources to data reduction and analysis on a real-time basis. Another popular interface method is to use an external communications link to the host computer. The advantage of this method is that the DAS hardware can be disconnected quickly and easily from the host, simplifying maintenance and increasing portability. The choice of DAS is often determined by issues such as cost, performance, portability and

ease of maintenance. Table 2.1 lists some of the available options and their main advantages and disadvantages.

In a DAS architecture, the conversion of electric signals to digital data is accomplished by a data converter. Conversion of a signal from the analogue to the digital domains requires a number of discrete and sequential operations to be performed. The control logic sends commands to the Analogue/Digital converter and reads the results. More sophisticated DA cards add functions that increase the ability of the device to behave autonomously from the host. One such function is the ability of a card to periodically acquire data and stream it to the host computer's memory without host intervention. Specialized DAS systems may also contain on-board digital signal processors (DSPs) as well as logic for capturing and processing specialized inputs such as video signals from video cameras.

Figure 2.6 shows the layer structure of a typical data acquisition system. Data obtained from the monitoring system is at the bottom of the architecture. Alarms function ensures the continuity and security of data transmission. The network interface (NI) forms the application interfaces (API), which provide a link between the network layer protocol and the actual application to implement network functionality. The network interface has to meet the following requirements: (i) Network layer protocol transparency. The NI should be independent of the network layer protocol. The functional calls in a NI should not be dependent on the underlying network layer protocols like TCP/IP, Novell's IPX/SPX, NetBEUI, AppleTalk, and ISO TP/4. (ii) Asynchronous operation. The NI function calls should not wait for a network action to occur and then continue further execution. Those types of function calls are known as blocking calls.

The blocking calls are not suited for preemptive multitasking operating systems (especially with a single processor). The function calls should respond to occurrence of an event. Those types of function calls are known as non-blocking calls. 3) Data transfer rate. The NI should have sufficient data transfer rate capability so that it does not cause the underlying network layer protocol to introduce delays.

The database management function is used to facilitate the manipulation and storage of data. Software architecture provides a tool to users to interact with the data communications. The most popular programming languages used for data acquisition are Microsoft Visual C++ and JAVA. Microsoft Visual C++ provides an Object-Oriented library known as Microsoft Foundation Class (MFC) to implement the WinSock interface. The applications developed are platform dependent (Windows 95 or NT). At present, this tool has wide acceptance in the software community. Java also comes with classes to implement the WinSock interface. The main advantage in using Java is platform independence. Java is also suited for embedded programming, which will help in developing embedded data acquisition systems consisting of Internet communication facilities.

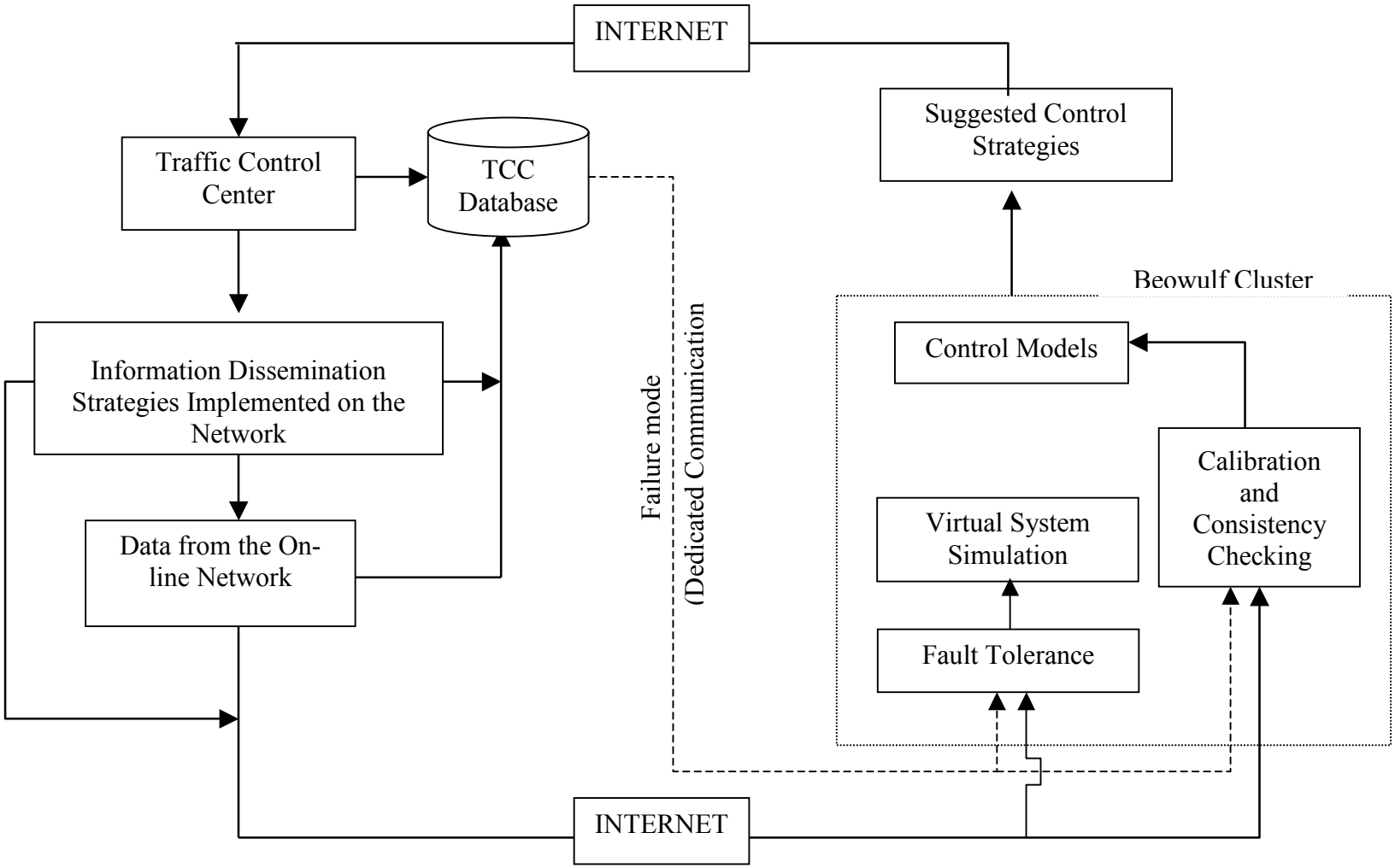


Figure 2.1 On-line Architecture for Real-Time Traffic Systems Control

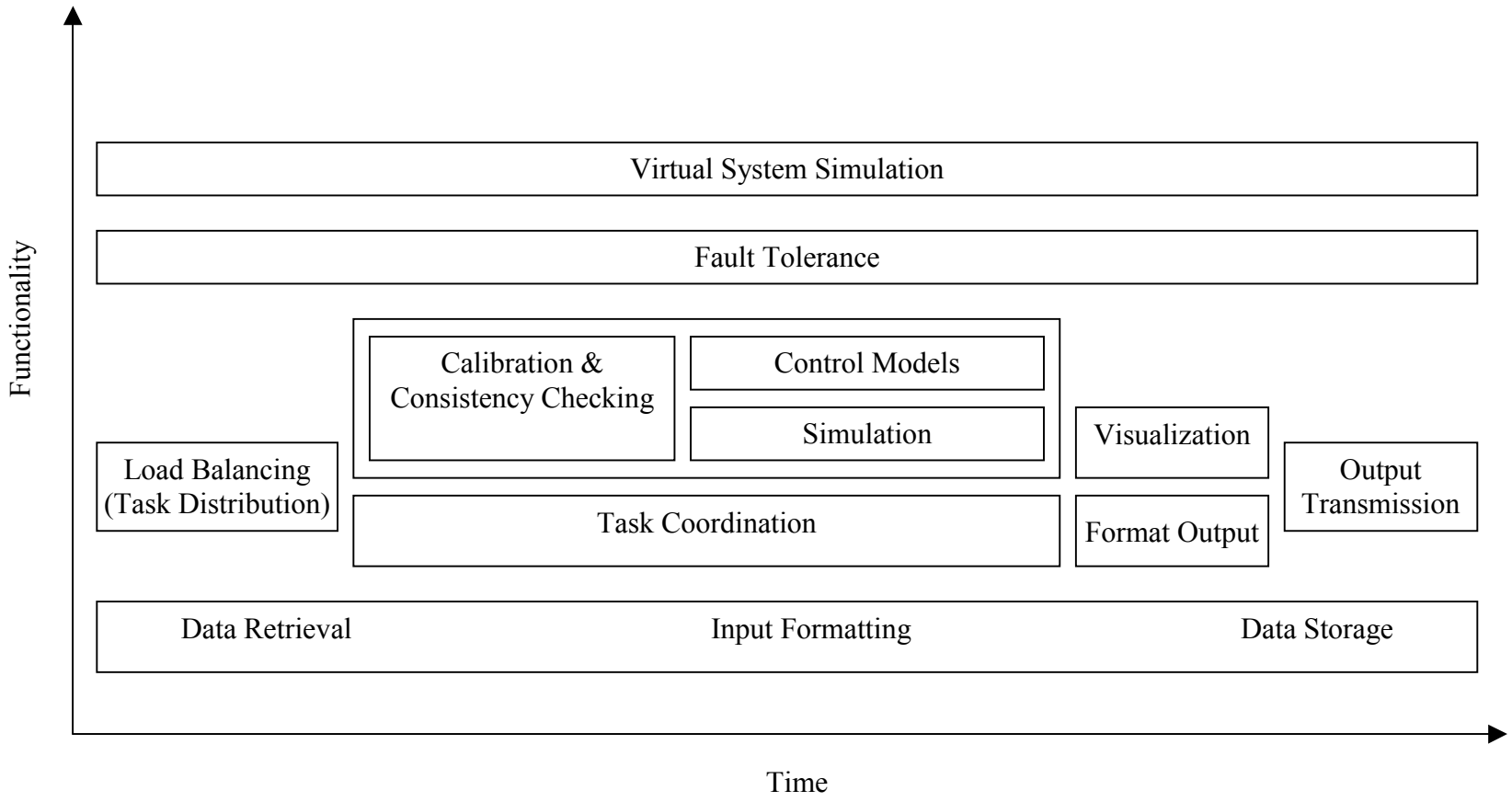


Figure 2.2 Concurrency Diagram

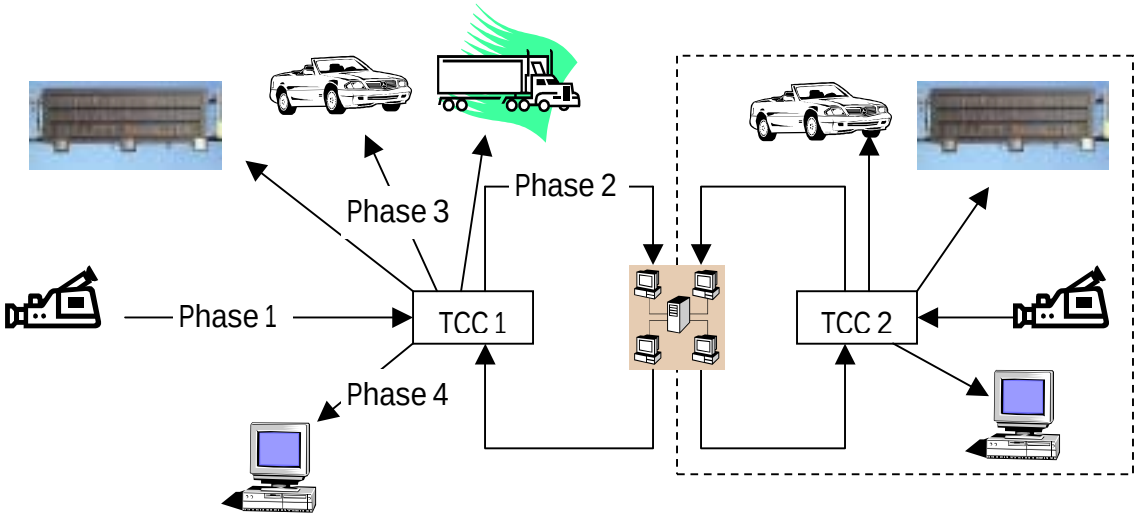


Figure 2.3 Data Communication Phases in On-line Traffic Control Architecture

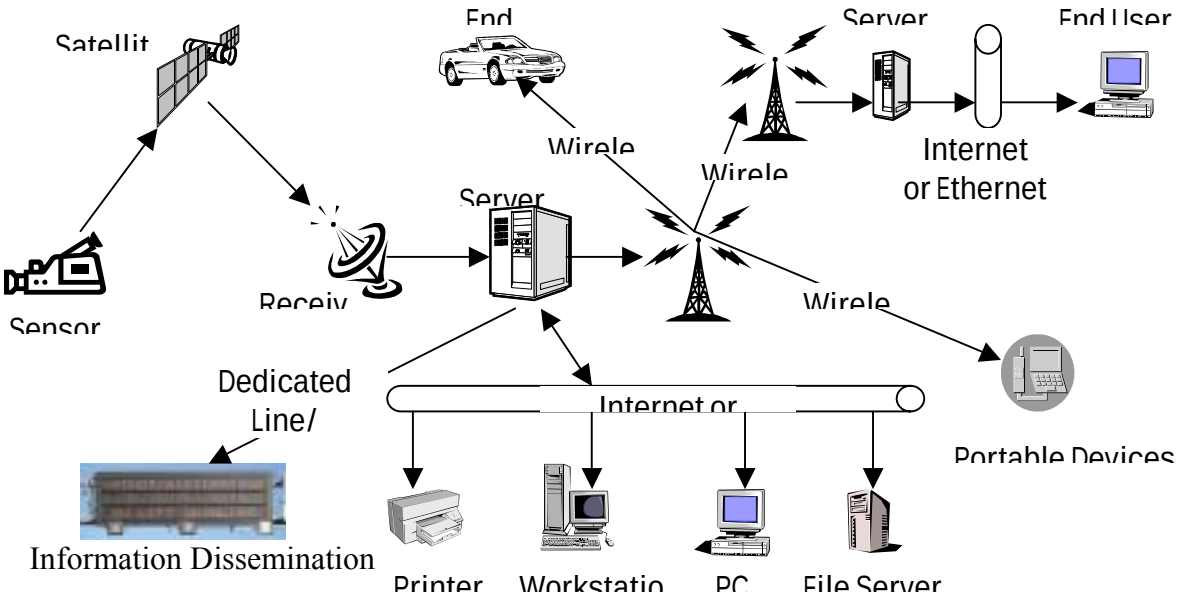


Figure 2.4 Telemetry of Data Acquisition Architecture

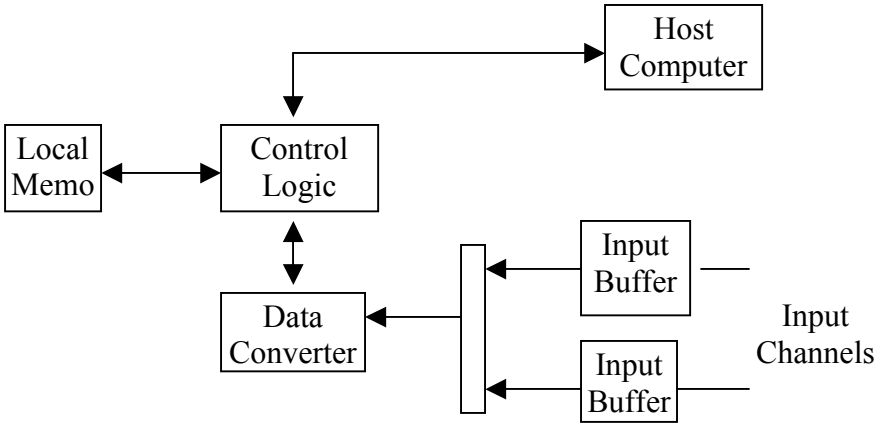


Figure 2.5 Data Acquisition System (DAS)

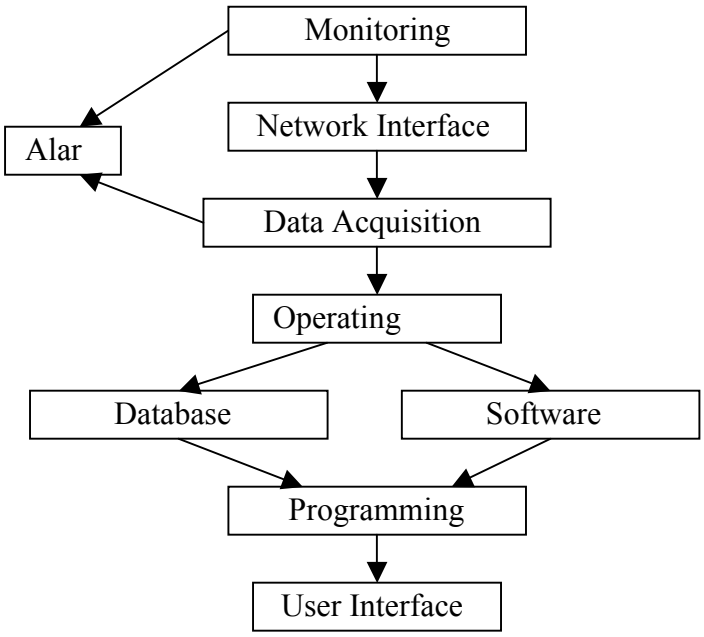


Figure 2.6 Layer Structure of Data Acquisition System

Interface/bus	Advantages	Disadvantages	Bandwidth (Mb/S)
ISA/EISA	(i) Fast (ii) Variety	(i) Obsolescent	1 (d-bit ISA) 33 (EISA)
PCI	(i) Very fast (ii) Plug-and-play	(i) Few cards available	133
RS-232	(i) Common	(i) Very slow	0.001-0.01
RS-485	(i) Fast	(i) Expensive	0.01-1
IEEE_488	(i) Fast	(i) Expensive	0.1-1
USB	(i) Fast	(i) Few cards available	0.2-1.5
Port IEEE-1284	(i) Fast	(i) Limited range	0.1-1

Courtesy: Adapted from Ramsden (1999)

Table 2.1 Comparison of DAS-to-host System Interfaces

3. BEOWULF CLUSTERS

3.1 The Beowulf Computing Paradigm

A Beowulf Cluster [1] is a collection of personal computers (PCs) interconnected using fast and dedicated Ethernet technology. PCs are extremely economical due to the mass production of microprocessors, memory chips, input/output (I/O) controllers, motherboard chip sets and the associated systems in which these components are incorporated. Further, the cluster typically uses one of several open-source operating systems such as Linux or FreeBSD which are freely available from the world wide web. Consequently, Beowulf Cluster costs are much lower than corresponding custom architectures that provide comparable computing power. Though the custom architectures (also called massively parallel processors or MPPs) use similar processors and memory, they have the burden of incorporating special communication features, packaging, and advanced compiler technologies that greatly increase cost and development time. Beowulf Clusters have the added advantage of flexibility since any obsolete components in the cluster can easily be replaced with minimal cost and time unlike customized architectures. Additionally, the Beowulf Cluster is flexible implying that the different computing units within its distributed architecture can have different computing speeds and memory capabilities. This flexibility provides the Cluster the capability to customize itself to the specific problem being addressed, and hence fosters efficient utilization. Since Beowulf Clusters use only mass-market components they are not usually subject to delays associated with custom parts and custom integration.

The primary philosophy of a Beowulf Cluster is to achieve explosive computational power at affordable costs. There are a number of inherent features that help the Beowulf Cluster achieve this objective. First, the PCs in the cluster are usually located in close proximity and are dedicated to executing a single application or program. This is the Beowulf Cluster's primary difference from a cluster of workstations (COW) where the PCs may be used for diverse tasks. For the same reason, the network experiences no external traffic thereby improving communication speeds between machines in the cluster. Second, the connections between machines in a Beowulf Cluster are through high-speed Ethernet adapters and switches that help reduce the inter-process communication times during the execution of different processes. Third, there is no theoretical limit to the number of PCs that can reside on a Beowulf Cluster. This implies that Beowulf systems can be expanded over time as additional resources become available or extended requirements drive system size upward. It should be noted however that more processors in a cluster does not necessarily mean faster execution times. This issue will be discussed in further detail in a later section.

The applications being executed on a Beowulf Cluster are typically programmed in languages such as C or Fortran which are compatible with traditional parallel programming interfaces. Parallel programming is a process by which an application is broken down into parts that can be executed simultaneously. Such parts are termed *concurrent. Parallel* parts of a program are concurrent parts that are distributed across the different machines for simultaneous execution. The different concurrent parts may sometimes require heavy data exchange during execution leading to high inter-process communication times. Critical to the success of the Beowulf computing paradigm is the

task of determining which concurrent parts of the program should be executed in parallel keeping in mind the possible trade-offs between inter-process communication times and computation speed-ups. The components resident in the cluster can then be customized for the optimal execution of the problem being addressed.

3.2 Architecture of the Beowulf Cluster

Figure 3.1 shows the architecture of a four node (five processor) Beowulf Cluster. Each of the components and the flow logic is discussed below.

3.2.1 Hardware

Node: The fundamental building block in clustered systems is referred to as a node which is a stand-alone computing system (a physical machine) with complete hardware and operating system support, and capable of execution of a user program and interaction with other nodes over a network. A node may contain one or more processors, memory, and hard disks along with other secondary components such as floppy drives, CD-ROM drives and Ethernet Cards. A node can access other nodes in the cluster through a fast switched-ethernet connection for the purpose of data transfer or retrieval. There is usually one *master* node and several *slave* nodes in a Beowulf Cluster. The master node is responsible for distributing tasks to the slave nodes in the Cluster according to a pre-specified parallelization scheme. The slave nodes perform the duties assigned to them and report back to the master node upon completion of the tasks. Any results reported by the slave nodes are then gathered and formatted by the master node for further processing. The master node is usually the only gateway implying that external access to any of the slave nodes can be achieved only through the master node. More complex

Beowulf Clusters may incorporate several subsystems with many master nodes each containing a group of slave nodes, interconnected by means of switches.

Processor: Processor family (Intel x86, IBM/Motorola PowerPC, DEC Alpha), clock rate of processor, and the cache size are the main characteristics associated with the processor. One or more processors can reside on each node of the cluster. In an Intelx86 dual processor machine, the main memory, which is shared between processors, can be used for communication purposes. This type of communication is much faster than communication between processors across nodes which uses the ethernet connection. For the same reason, it may be desirable to have as many processors as possible within the same physical machine.

Memory: The performance of a Beowulf Cluster is as much dependent on the memory subsystem as the processor. The amount of memory available can be extremely critical to the execution times of the application. During the data input stage of an executing program, a processor sometimes reads a large data file and stores it onto the local memory for subsequent use in a program. If adequate memory is unavailable, this data is read from the hard disk as and when necessary, a procedure that is much slower than data retrieval from memory. In such a case, adding more memory to a node or distributing the task across nodes should be considered. Memory chips are easily installed on processor motherboards and may be upgraded at any time.

Hard Disk: Hard disks not only provide non-volatile storage but also provide a means of extending the apparent memory capacity either automatically through virtual memory or explicitly through programmer control of the file system. Through a judicious choice of

hard disk size for each node, one can ensure efficient data transmission in the Beowulf Cluster. This issue is discussed in greater detail in a later section.

Switch: The demand for higher sustained bandwidths and the need to include a large number of nodes on a single network have spurred the development of sophisticated means of exchanging messages among nodes. Switches, like hubs and repeaters, accept packets on twisted-pair wires from the nodes. Unlike repeaters, these signals are not broadcast to all connected nodes. Instead, the destination address fields of the message packets are interpreted and the packet is sent only to the target node or nodes.

3.2.2 Support Devices

A portable 1.4MB floppy disk drive with form factor 3.5" is still used to some degree, but with the advent of inexpensive CD-ROM and the availability of the Internet, it is fast losing its preeminence. Floppy drives are crucial for system installation and crash recovery. CD-ROM is an optical storage medium, and has become the principal medium for distribution of large software packages. While not necessary, most Beowulf Clusters have a direct user interface. One of the nodes (usually the master node) is used as a host and is connected to the monitor, keyboard, and mouse interfaces. This node is employed primarily for system administration, diagnostics, and presentation. The Network Interface Cards (NIC) provide communication access to the nodes' external environment. Each of the slave nodes contains a NIC that connect to the switch to facilitate communication within the Beowulf Cluster. A second NIC on the master node provides a link between the Beowulf Cluster and the local area network (LAN) which connects to other resources such as file servers, terminals, and the Internet.

3.2.3 Software and Data Issues

Operating System: The operating system grants access to the processor and memory, provides services to the application programs, presents an interface to the end user, and manages the external interfaces to devices. Beowulf Clusters typically use an open source operating system such as Linux where all the source code is freely available for modifications and improvements. Linux, in particular, which is a multitasking, virtual memory, POSIX compliant operating system, supports the complete GNU programming environment and is the most popular operating system for Beowulf Clusters. Several commercial versions of Linux (e.g. RedHat) are now available that allow for easy installation and use.

Network File System: The Network File System (NFS) provides an environment for seamless data access and system administration on the Beowulf Cluster. NFS facilitates remote hard disk access to each of the nodes in the Beowulf Cluster. Disk access through NFS is slightly slower than data transmission through message passing and hence NFS has generally been used only for system administration. It can be used to keep a single record (usually in the master node) of all accounts in the Cluster to facilitate easy maintenance of account information. NFS is known to open up a number of security issues which should be addressed before installation. As an alternative, especially for small and medium-sized Beowulf Clusters, account management can be performed by scheduling frequent transfer of account information from the master node to the slave nodes using commands such as *rdist*. Though this method is more secure than NFS, it may be unsuitable for large clusters (more than 64 nodes) due to the associated increase

in network traffic.

Data Storage and Transmission Issues: An issue that is critical to the efficient functioning of the Beowulf Cluster is the data storage and transmission mechanism that is used. An application that is being executed often requires multiple I/O operations on files at various times in the execution cycle. In a Beowulf Cluster, where some or all of the constituent nodes have their own hard drive, the efficiency of these I/O operations become critically dependent on the location and the manner in which files are stored and transmitted. The user typically has a number of options due to the flexibility of the Beowulf architecture. In Figure 3.2, three different data storage and transmission techniques are depicted. As is often the case with a Beowulf Cluster, there is no single most efficient method of data storage and transmission. The efficiency of any particular scheme will often depend on the (i) size of files being stored, (ii) variance in the file sizes, (iii) network speed, (iv) number of nodes and (v) the nature of file operations. For example, in Figure 3.2(a) separate copies of all files are stored on each node so that each processor in the cluster has free access to all the data from its own hard drive. Problems with this data storage scheme are the associated increase in data storage requirements, the need for frequent synchronization of data across nodes and a more convoluted software coding procedure. However, it was observed through simple experiments that the method performs well for small applications where file synchronization is relatively simple and file sizes are so small that any parallel implementation of I/O operations provides only marginal benefits. By contrast, in Figure 3.2(b) a single processor is fully responsible for data storage, subsequent retrieval and transmission to other nodes. Although such a mechanism is straightforward to implement, several processors are typically idle during

the data retrieval phase thereby causing inefficiencies. In addition, this scheme will be efficient only in the presence of a network with high data transmission rates. Scheme 3 shown in Figure 3.2(c) uses each processor to read a portion of the data and the processors subsequently exchange their data using message passing techniques. There are some issues to be considered when choosing between Scheme 2 and Scheme 3. If the files are uniform in size and the cluster in use is small to medium-sized (64 nodes), Scheme 3 may be more efficient than Scheme 2. This is because, uniformity in file sizes ensures that parallelization of I/O operations is a relatively simple procedure and does not involve elaborate load balancing techniques. Furthermore, a small number of nodes in the Beowulf Cluster makes sure that network traffic does not clog the switch due to the large number of messages that are being passed between the nodes. If either the files are heterogenous and/or the cluster is large, a hybrid scheme where a group of processors are dedicated to data retrieval, storage and transmission may prove the most effective. All of the three schemes discussed can also be implemented alongside NFS. NFS obviates any need for synchronization of files across nodes since each node has full access to a dedicated location where all data is stored. The obvious issues associated with using NFS for data storage are the associated increase in retrieval times and the need for a single large storage device. The former issue can be circumvented in most cases by a careful tuning of network settings soon after installation of NFS. In particular, the efficiency of NFS depends heavily on the size of data blocks that are used by the server and the clients (rsize and wsize) in NFS. With careful testing and optimization of these parameters, any overhead due to NFS can often be made marginal. Also, the drastic reduction in the price of storage devices is making the requirement for a large and fast storage device for NFS

much less of a concern. An alternative to manual optimization of data storage and transmission mechanisms is the use of dynamic load balancing software such as MOSIX. MOSIX with its own file system and the ability to constantly monitor the loads and the available memory on each node, can seamlessly migrate tasks across nodes in the cluster so as to dynamically balance loads. This feature combined with its fault tolerance capability where processes on a dysfunctional node are automatically migrated to other nodes can make MOSIX a valuable tool in the efficient operation of Beowulf Clusters.

Message Passing: Message passing is the mechanism by which data and instructions are conveyed between parallel executing processes in a distributed computing system [2]. In other words, message passing facilitates the precise assignment of tasks to various processors and any exchange of data during execution. An efficient message passing mechanism is extremely crucial for fast execution of parallel programs especially when the application contains highly interdependent parallel tasks. In such cases, there is a large amount of inter-process communication which may be extremely costly vis-à-vis execution time. In the current context, the message passing function is performed using an implementation of the Message Passing Interface (MPI) called MPICH. Since MPI is an open standard, there are several public domain implementations that are freely available. An important advantage of MPI is its portability. Programs written in MPI for one architecture can be compiled and executed on other architectures without any modification. This feature is especially useful for Beowulf Clusters which may contain nodes that frequently change in architecture and environment. The other popular message passing tool is called Parallel Virtual Machine (PVM) but unlike MPI, PVM is not an open standard.

3.3 Super-computing Cluster for On-line and Real-time Control of Highways using Information Technology (SCORCH-IT)

We currently have installed a 16-processor Beowulf Cluster, called SCORCH-IT, on which the on-line control architecture for real-time route guidance will be implemented. SCORCH-IT was recently expanded from four processors to sixteen processors based on insights from a set of experiments using a time-dependent shortest path algorithm. Since the cluster is constructed only from commodity hardware and runs open source software, the system costs are significantly lower than those of customized supercomputers that do not scale well economically.

SCORCH-IT consists of eight rack-mount cases, each of which constitutes one node (a physical machine) of the cluster. Each node consists of an ASUS-P2BD motherboard with dual Pentium III 500MHz or 550MHz processors and a 256MB SDRAM memory card with provision for expansion. 3c905B network cards are installed on the eight nodes along with a Cisco 2900XL switch (with a back plane of 2.4GB) to form a switched Ethernet connection using 10/100BaseT Category 5 cables. The nodes are configured for a private Beowulf Cluster network and the front-end machine (Node0) has an additional network card with a real Internet Protocol (IP) address forming the only link to the outside world. Hence, the master node is not only on the local network but is also the gateway for the Intranet that it forms with the other nodes. The nodes have Linux kernel 2.2.5-15 installed and boot from their respective hard drives [3]. Account management using rdist keeps identical account information on all nodes. The parallel

computing software consists of the core MPI implementation [2], profiling libraries (MPE), and a visualization tool called Jumpshot.

3.4 A Sample Program Execution

The Multiple User Classes Time-Dependent Traffic Assignment (MUCTDTA) algorithm is a large-scale optimization solution algorithm developed for providing optimal routes to users on-line, for traffic networks with ATIS/ATMS capabilities. The algorithm solves for the time dependent assignment of vehicles to network paths for a given set of time-dependent origin-destination trip desires. As a first step, a component from this algorithm was chosen for parallelization to obtain insights on the potential speed-ups for real-time deployment using the 4-processor Beowulf Cluster paradigm.

An important stage in the MUCTDTA algorithm is the calculation of the shortest paths for every time step over a period of interest in a network with time-dependent arc costs [4]. In the current context, a time-dependent shortest travel time path algorithm (SPA) performs this function. SPA, as shown in Figure 3.3 consists of three distinct parts: (i) READ (ii) COMPUTE and (iii) OUTPUT. In READ, data on the network topology (fixed characteristics such as geometry), signal timings and traffic information (current travel times) is read from static files created by a simulation component called DYNASMART. The input data files are accessed sequentially and stored into large arrays in memory (RAM) to facilitate faster data retrieval. In order to account for the possibility of large input files caused by longer periods of interest and/or larger networks, adequate memory should be provided on each of the nodes in the cluster. Input/Output (I/O) is generally not suitable for parallelization in computationally intensive

applications. However, in I/O intensive modules, significant savings in execution time may be obtained by assigning the available processors to read different portions of the same input file and then exchanging the accessed data through memory. The second part (COMPUTE) consists of the time-dependent shortest path calculation for a given destination node from all nodes in the network. This step is repeated for each of the discretized time intervals and for each destination in the period of interest. The iterations by destination do not share or exchange results and hence are amenable to parallelization. Such modules that contain independent and computationally intensive loops are termed *embarassingly parallel* in programming literature, signifying the ready gains obtainable through parallelization. Results from the SPA procedure are written to separate output files for each destination in the network by the third component (OUTPUT). This component is also readily parallelizable by destination since there is no post-processing of files that needs to be performed in SPA.

Experiments were conducted under different parallelization schemes using the four processors in SCORCH-IT. The objective of these experiments was to decrease total execution time and also gain insights into the scalability of SCORCH-IT. For instance, the results obtained may suggest investment in processors as opposed to memory, or increasing network speeds as opposed to processors, etc. Some important performance measures used in this analysis include: (i) *Total Time:* This refers to the total time of execution of the program. It includes time taken for computation, inter-process communication, and input/output. When averaged over several runs to account for system randomness, it is called *Average Total Time*; (ii) *Computational Time*: Computational time is the time spent only on computing. When averaged over several runs, it is called

Average Computational Time; and (iii) *Data Input Time*: This refers to the time spent in reading data files from the hard disk. When averaged over several runs, it is called *Average Data Input Time*; (iv) *Data Output Time*: This measure refers to the time spent in writing output data files onto the hard disk. When averaged over several runs, it is called *Average Data Output Time*; (v) *Speedup*: The speedup indicates how much faster a problem will be solved using parallelization as opposed to a single processor [5]. Speedup, denoted by S(P) is defined as the ratio T(1)/T(P) where T(1) refers to the Average Total Time on a single processor and T(P) refers to the Average Total Time on a system with *P* processors. Speedup is an important performance measure used to determine the performance level of the Cluster for the current execution; (vi) *Efficiency*: This measures the fraction of the time a typical processor is busy. Efficiency, denoted by E(P) is measured as the ratio S(P)/P. Ideally, E(p)=1, implying that no processor remained idle during the execution.

3.4.1 Test Network Description

The test network that was used in the experiments consists of 178 nodes, 441 links, 20 origins and 20 destinations. A bi-directional freeway runs through the entire length of the network with the urban street network consisting of arterials and local roads on both sides of the freeway. The free-flow speed on the freeway links is 65 mph, and 40 mph for most other links in the network.

3.4.2 The Control Variables

The following parameters were used as control variables in the experiments that were performed using SPA:

Period of Interest. The period of interest (T) refers to the time interval during which time-dependent shortest paths are desired. The experiments in this study use a range of T values between 10 minutes and 100 minutes.

Load Factor. This parameter is a measure of the congestion level in the network. The load factor is defined as the ratio of the total number of vehicles generated during the period of interest compared with a base number of 16717 vehicles (over the same period) which corresponds to a load factor =1.00. In all of the experiments performed using SPA, the load factor was held constant at 1.00.

Loading Profile. This parameter refers to the time-dependent pattern in which vehicles are loaded onto the network. For instance, a uniform loading profile refers to a fixed number of vehicles being loaded during each time step. Though the loading profile and the load factor do not directly affect SPA, these factors in combination with the network topology decide the time-dependent link travel times which in turn influence the execution time of SPA. The loading profile was held constant for all experiments in the study.

Number of Processors. This refers to the number of processors P, used to execute the SPA module.

3.4.3 Description of Experiments and Results

SPA was separated from the MUCTDTA algorithm and executed separately using input files generated externally. Table 3.1 describes the different experiments that were performed on SCORCH-IT. Each experiment was performed several times to account for variations in processor speeds, data retrieval and output rates.

Experiment 1: In Experiment 1, the sequential version of SPA was executed on different processor architectures and operating systems using a single processor for a 30 minute period of interest. This step was aimed at gaining insights on the relative performance of each of these environments. Table 3.2 summarizes the results from this stage of the experiment. As can be seen from the table, the execution times are ordered according to the processor speeds. Faster processors outperform the slower processors irrespective of the operating system in use. In the case of COMPUTE, this is because of the large number of numerical calculations involved. Any differences due to the operating system are more than offset by the effects of processor speed. In each of the cases presented, the computational time comprises of more than 95% of the total time of execution. Therefore, the trends observed with the total time are similar to what was observed with the computational time.

Experiment 2: In Experiment 2, the parallel version of SPA was executed on SCORCH-IT using different numbers of processors. The individual execution times of each of the three parts READ, COMPUTE and OUTPUT were recorded and analyzed. In this context, a process is defined as a task that is spawned on a processor. Figure 3.4 shows the parallel implementation of SPA across four processors with one process executing on each processor. In the first step, each of the four processes reads input data files and stores them into arrays in the memory of the local machine. Next, each process is assigned a destination for computation of time-dependent shortest paths. On completion of this task, results are written to an output file on the local hard disk. This procedure is repeated until all destinations in the network have been distributed. For example, in a network with 16 destinations, such a parallelization method would result in 4 destinations

being assigned to each processor. This is not necessarily the best parallelization scheme because, depending on the nature of destinations assigned, some processors may finish faster than others and remain idle during parts of the program execution. A more optimal parallelization scheme would be a dynamic allocation of tasks where tasks are assigned to processors as and when they become idle. This may result in an unequal distribution of destinations across processors but will ensure maximum efficiency. More recent software tools such as MOSIX facilitate such a dynamic load balancing procedure.

Figure 3.5 shows the relationship between the computational time and the number of processors used in Experiment 2. Since COMPUTE is embarassingly parallel, we would expect the execution time of this portion to be inversely proportional to the number of processors used (directly proportional to 1/P). This can be observed from the graph which resembles a K/P curve, where K is the execution time when COMPUTE is executed sequentially. Small deviations from this expected relationship were observed in the data and these may be attributed to the sequential portions of COMPUTE. Sequential portions within a program can be thought of as fixed-costs that cannot be reduced by parallelization. Figure 3.6 shows the same relationship for the READ and OUTPUT portions of SPA. READ was not subject to any parallelization and hence any variation that is observed is purely due to the natural randomness in the data retrieval rates. OUTPUT accounts for a tiny fraction of the total execution time but the trends exhibited are similar to those of COMPUTE.

Symmetric Multiprocessor Machines (SMP) are machines with multiple processors capable of sharing a single pool of memory. Such an architecture facilitates parallelization of the data retrieval process through a procedure called *Threading*. This

procedure facilitates communication through shared memory in SMP machines. Such a communication is generally much faster than through messaging (e.g. using MPI). Threading between processors across machines is a substantially more complex task and requires special hardware. Threading was not performed in SPA because the potential time savings through the process are marginal when inter-process communication accounts for only a small fraction of the total time. The lack of substantial inter-process communication in SPA allows for accurate modeling of the total time. Total time can be approximated using a model with two components: (i) a fixed component (K_1) that is constant across experiments and accounts for the execution time of READ and any sequential portions in COMPUTE and OUTPUT, (ii) a variable component (K_2/P) where K_2 is a constant. This component depends on the number of processors and accounts for the execution times of COMPUTE and OUTPUT. Figure 3.7 shows the results of a least squares regression that was performed using such a model form. The diamond and dashed markers represent the observed and the predicted average total times as a function of the number of processors used. The fixed component was estimated at 9.617 sec indicating that the minimum time of execution of SPA is 9.617 seconds and this is achieved at the theoretical limit of an infinite number of processors. It can be seen from the model that the time savings due to a unit increase in the number processors actually decreases with an increase in the existing number of processors. This is clearly evident through Figure 3.8 that plots the marginal time savings as a function of the number of processors. Here, the y-axis represents the time savings that can be achieved by adding one more processor to the cluster while the x-axis represents the current number of processors in the system. All of the measures discussed thus far are for a single execution of SPA. Multiple

iterations (usually 6 TO 7) of SPA are required for converged solutions in real-time and hence all absolute time savings have to be multiplied by this factor correspondingly.

Experiment 3: Experiments 1 and 2 were performed with a fixed period of interest of 30 minutes. Experiment 3 was designed to demonstrate the effectiveness of SCORCH-IT for longer periods of interest using the same load factor, loading profile and network in question. It should be noted that the objective of this experiment is not to study the sensitivity of SPA for different periods of interest, a task that would entail a more detailed probabilistic analysis of travel times in the network. Four different periods of interest (10minutes, 30 minutes, 60 minutes and 100 minutes) were chosen and SPA was executed first on a single processor and then using all four processors in SCORCH-IT. Figures 3.9 and 3.10 show the results from this experiment. It is evident from the figures that the average computation time increases non-linearly with respect to the length of the period of interest. Longer periods of interest entail an increase in the fraction of time spent on computation and the non-linear trend is due to the fact that SPA has a worst case complexity of $O(N^3T^2)$, where N represents the number of nodes [4]. This is advantageous from a parallelization standpoint since COMPUTE is embarrassingly parallel and results in speed-ups between 2.31 and 3.47 with increasing period of interest. The figures also demonstrate the increasing trend of I/O execution time with period of interest. Longer periods are associated with larger data files leading to a corresponding increase in the I/O time. Parallelization of I/O along with a more optimal placement of input data files depending on the functionality of the nodes in the cluster may result in increased speed-ups.

Some important conclusions can be made from the above experiments: (i) SPA is

amenable to parallelization providing significant savings in execution time. However, this observation may be true only up to around 10 processors after which additional processors provide negligible savings. (ii) Memory requirements for the execution of SPA are well within the existing resources in SCORCH-IT. However, this conclusion applies only to the specific network that was used in the experiments. Larger networks and/or longer periods of interest are associated with larger input data files. Hence, if swapping from the hard disk is to be avoided, additional memory may be needed in the nodes. Table 3.3 shows the speedup and the efficiency estimates for each of the different scenarios. The generally accepted lower limit of speed-up for high–performance computing is $P/2^5$. By this definition, it is evident from Table 3.3 that even when all four processors are used, the Cluster is still performing in the High-performance range. As we increase the number of processors in the system it can be expected that the Cluster may fall into the Minimum high-performance or the Intermediate performance ranges for the execution of SPA.

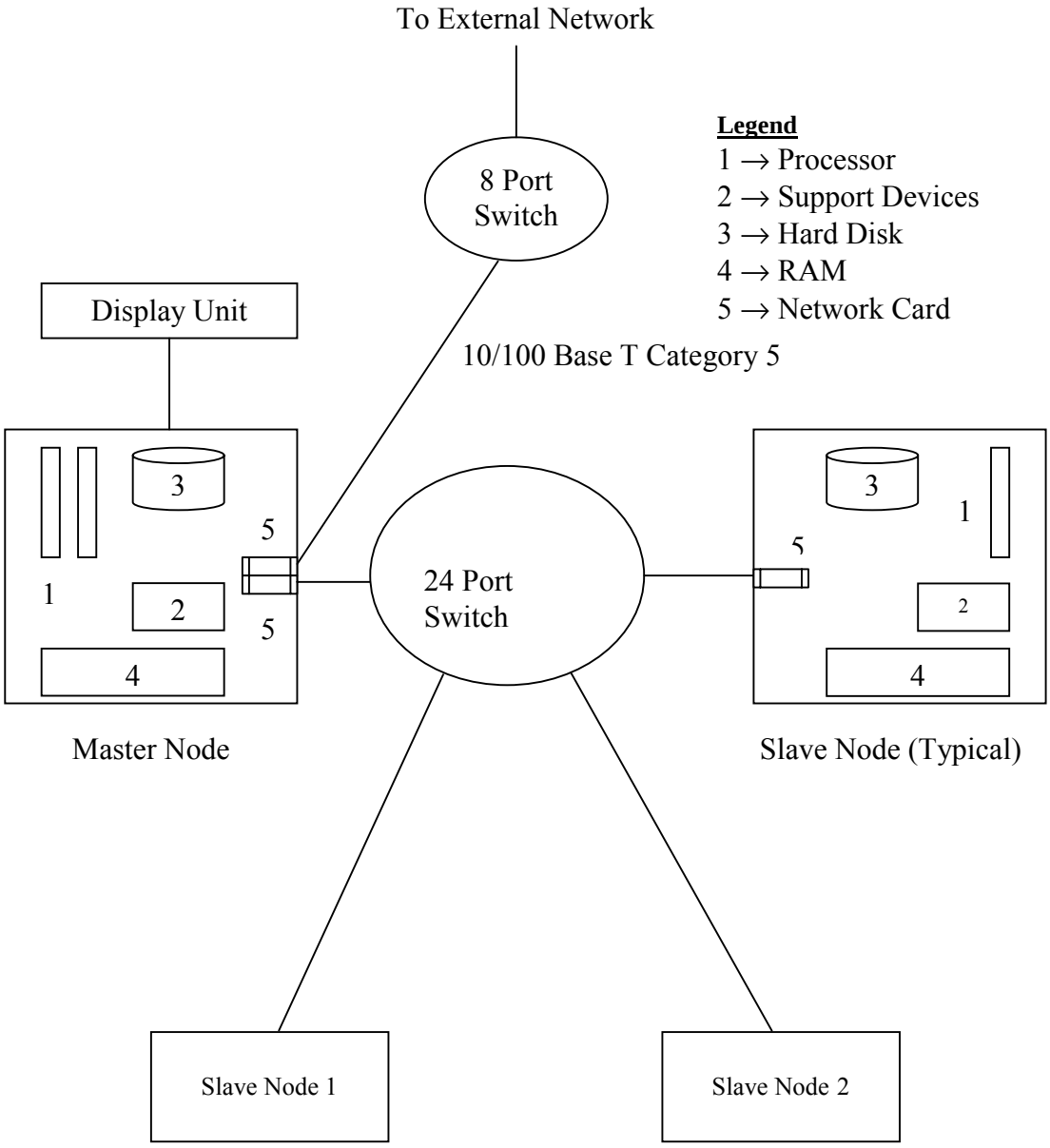


Figure 3.1 Beowulf Architecture

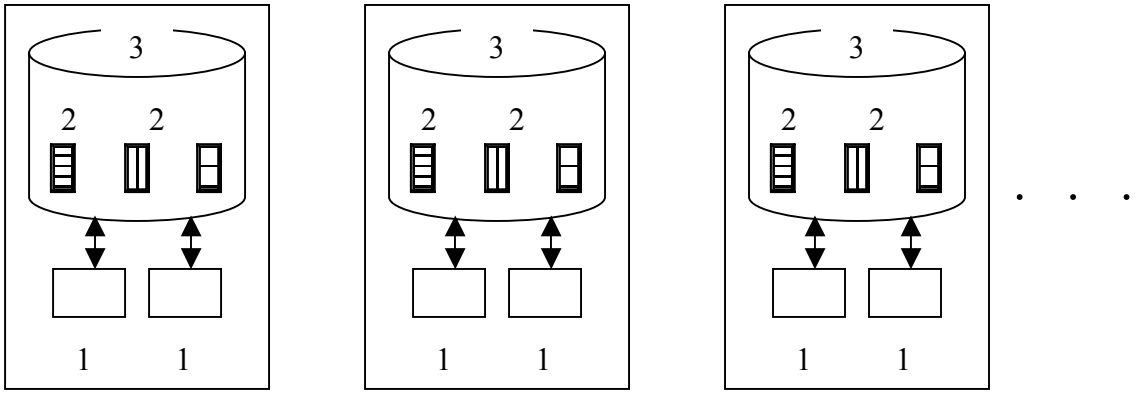


Figure 3.2(a) Data Storage and Transmission, Scheme 1

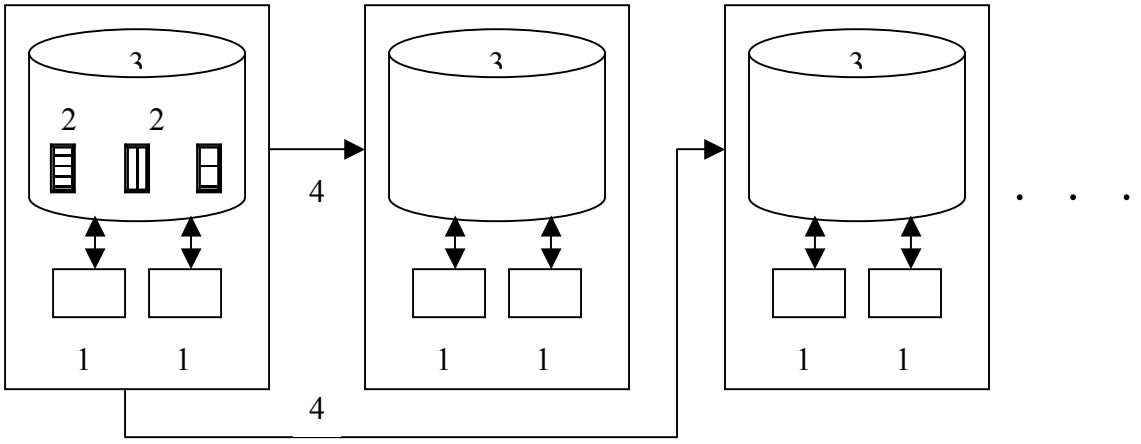


Figure 3.2(b) Data Storage and Transmission, Scheme 2

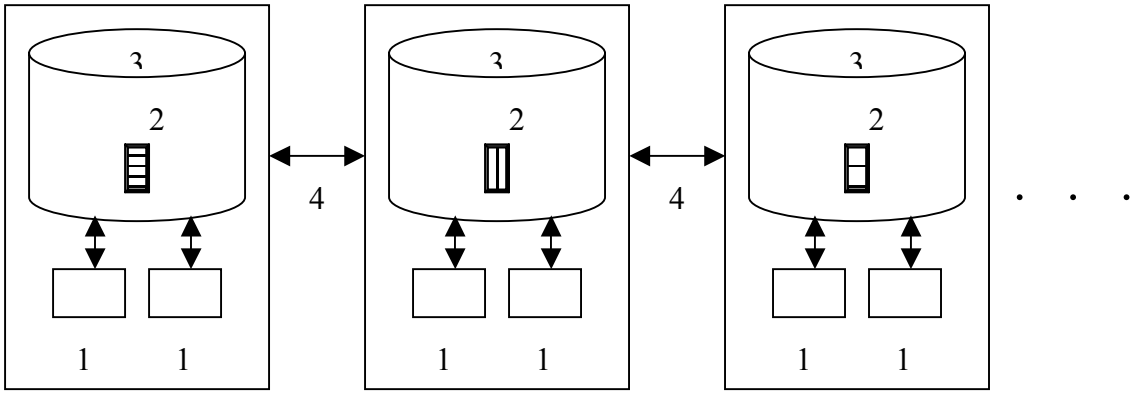


Figure 3.2(c) Data Storage and Transmission, Scheme 3

Legend

1 → Processor , 2 → File, 3 → Hard Drive, 4 → Message Passing

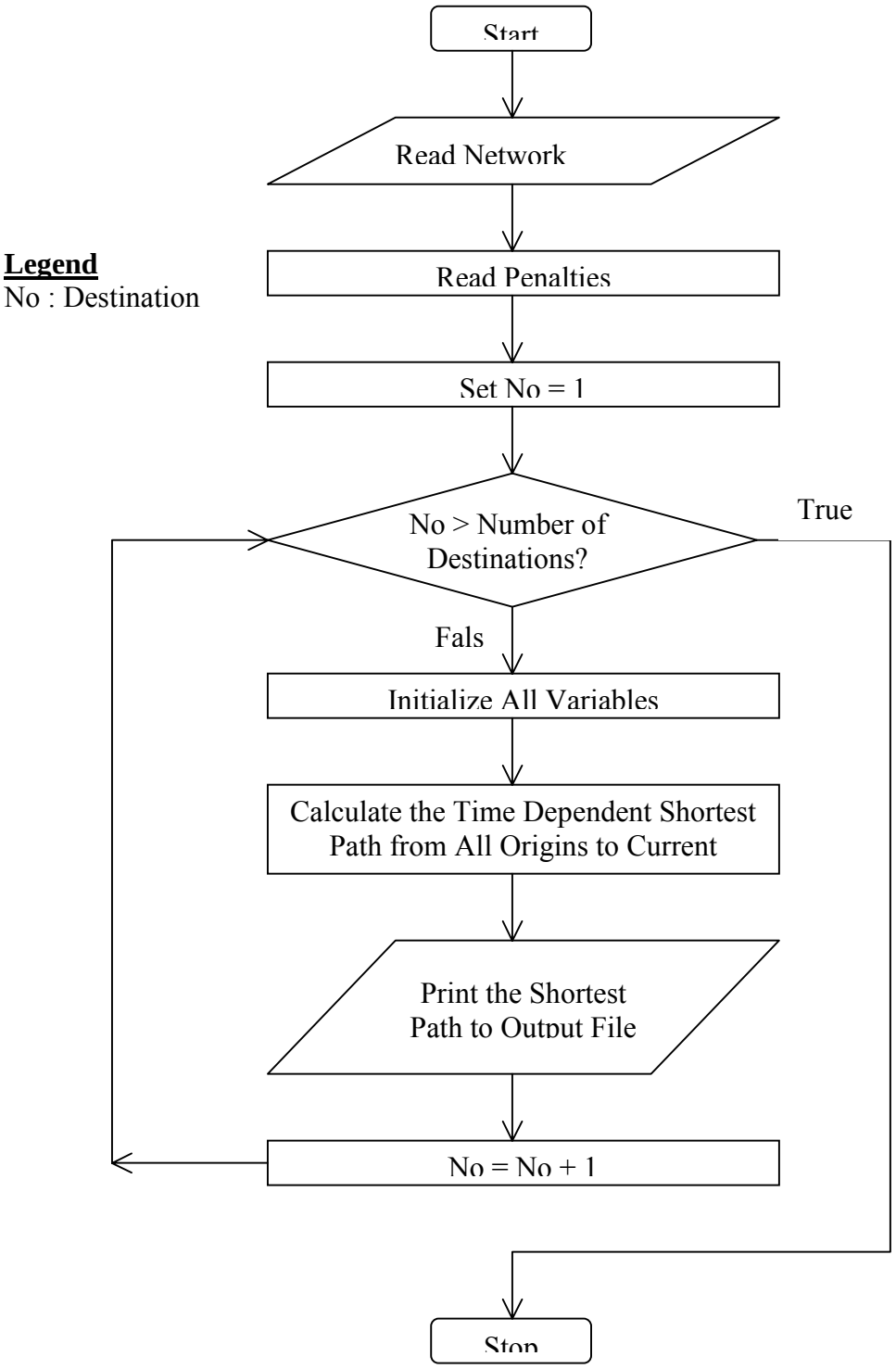


Figure 3.3 Time-Dependent Shortest Path Algorithm

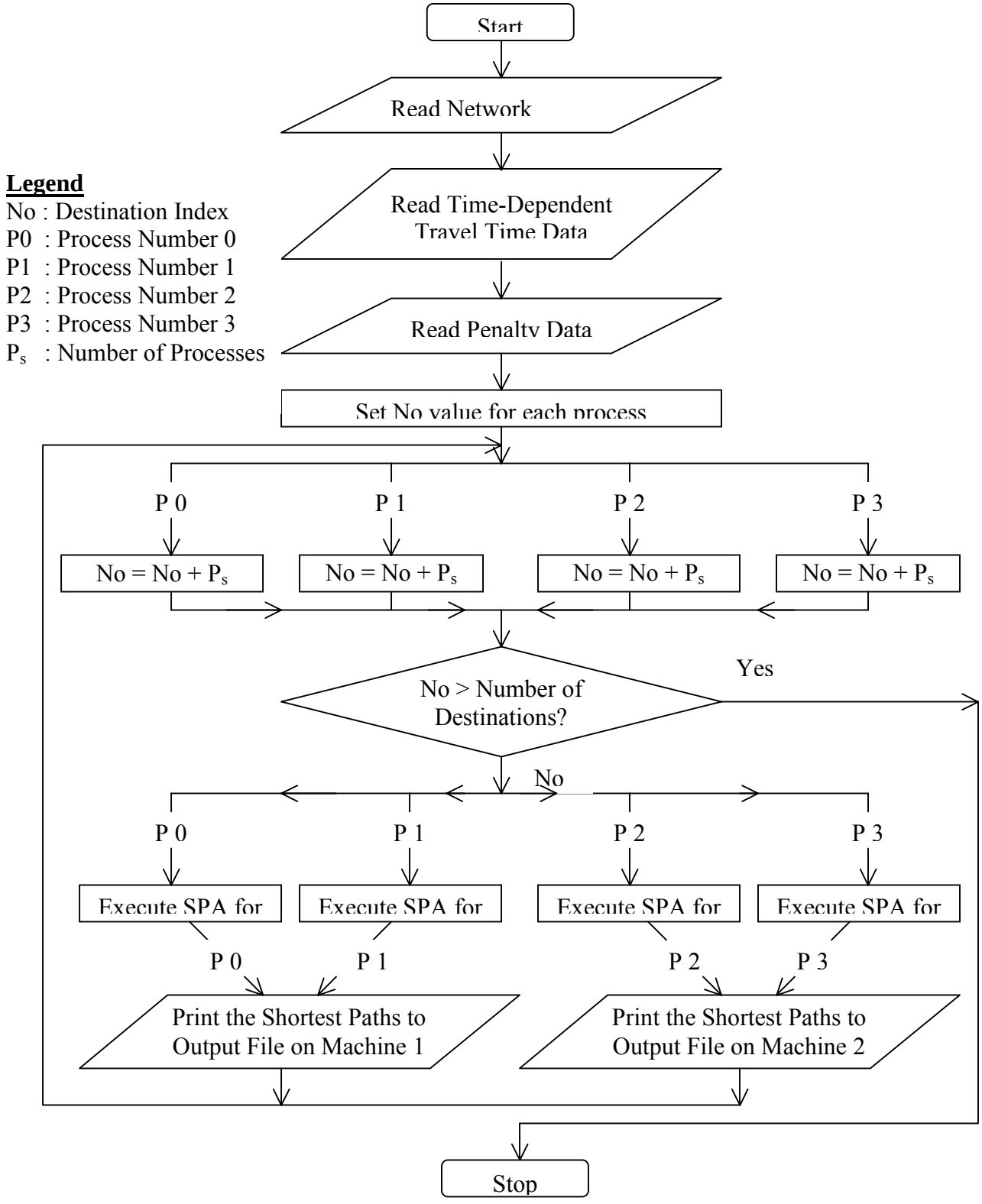


Figure 3.4 A Parallel Execution of the Time-Dependent Shortest Path Algorithm

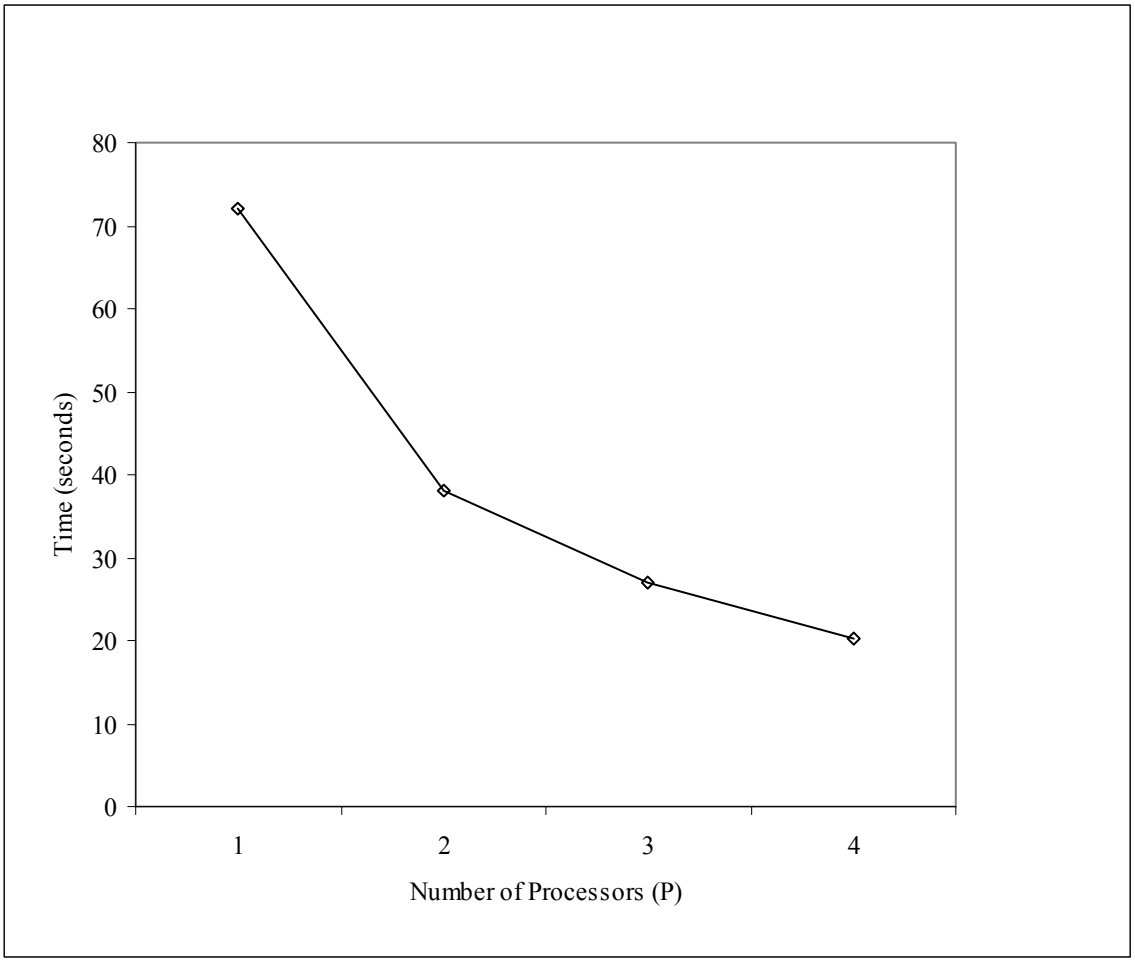


Figure 3.5 Average Computational Time (Experiment 2)

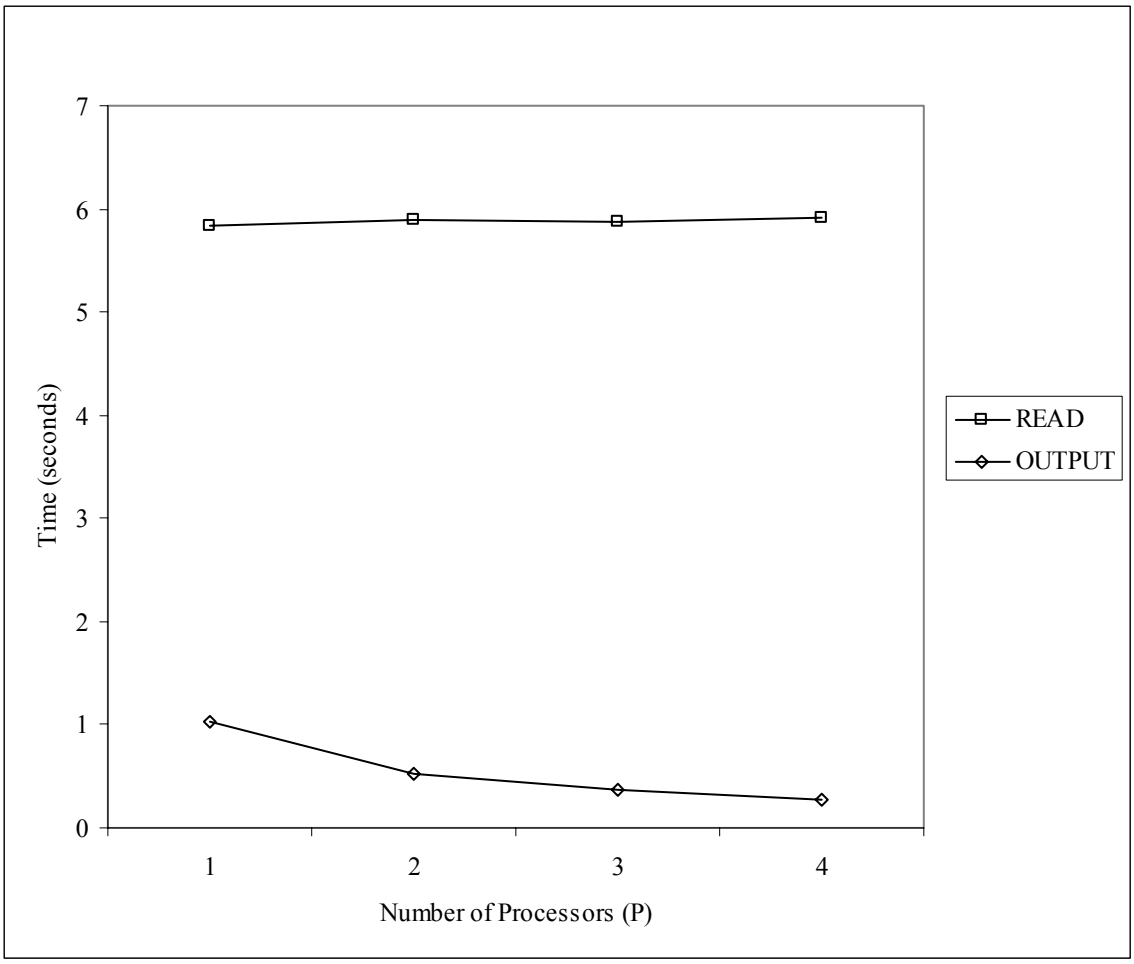


Figure 3.6 Average Data Input and Output Times (Experiment 2)

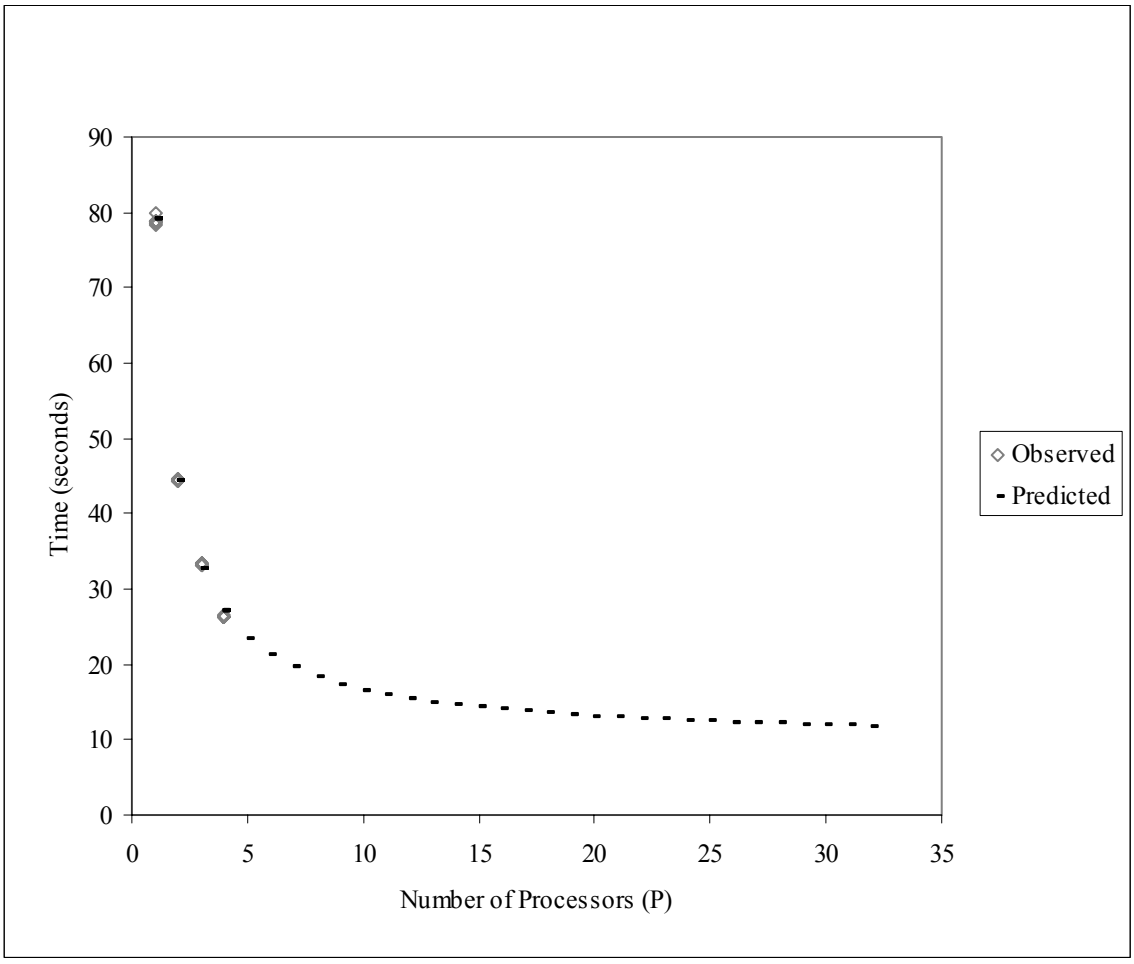


Figure 3.7 Prediction of Average Total Time (Experiment 2)

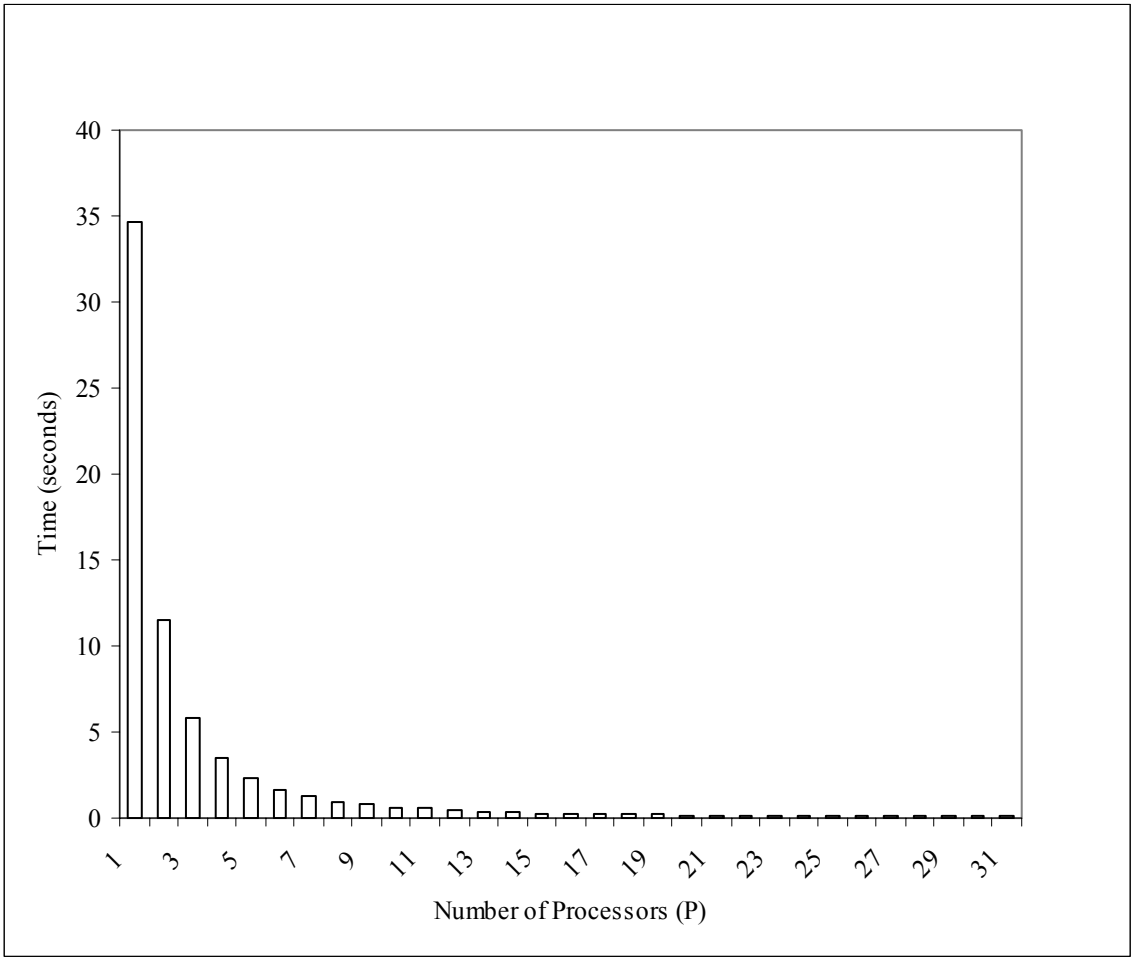


Figure 3.8 Marginal Time Savings (Experiment 2)

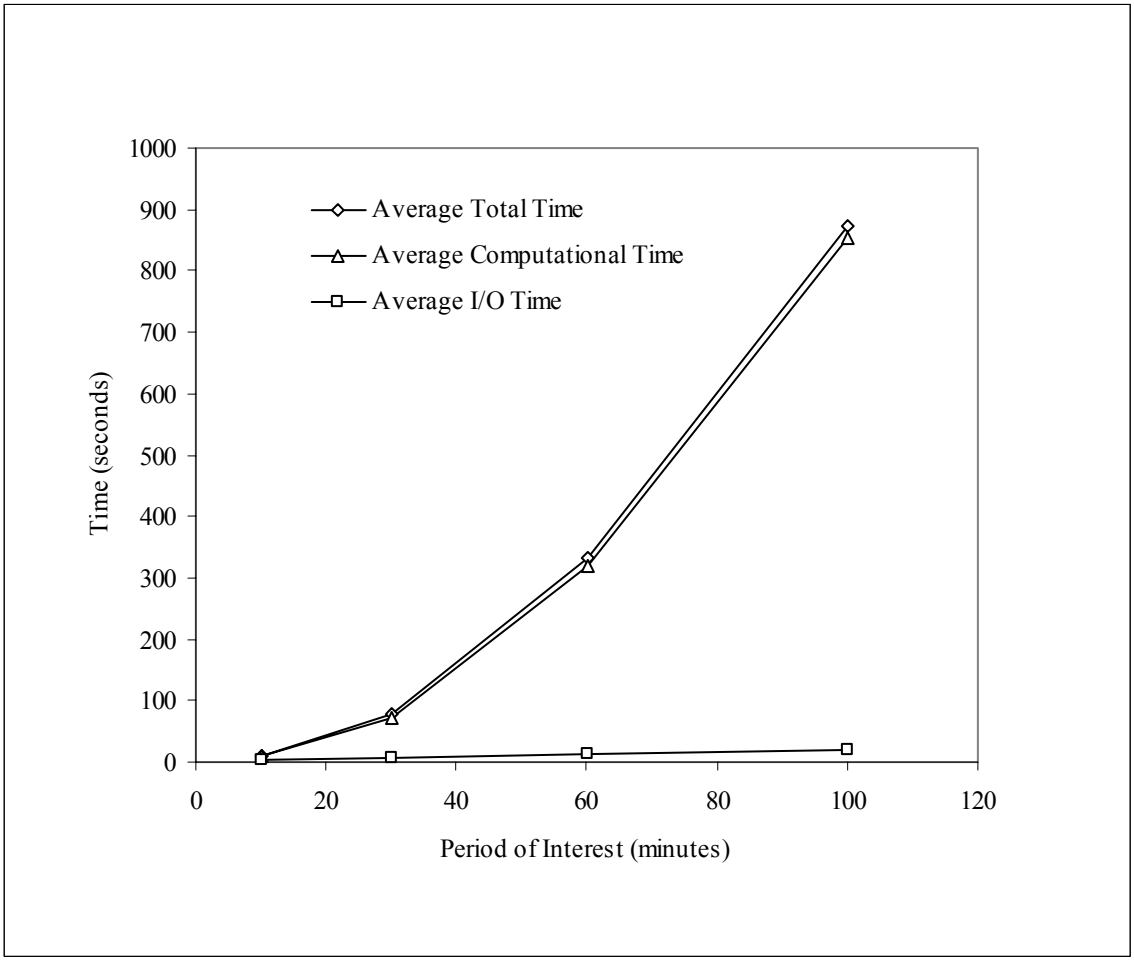


Figure 3.9. Execution Time by Parts (Sequential)

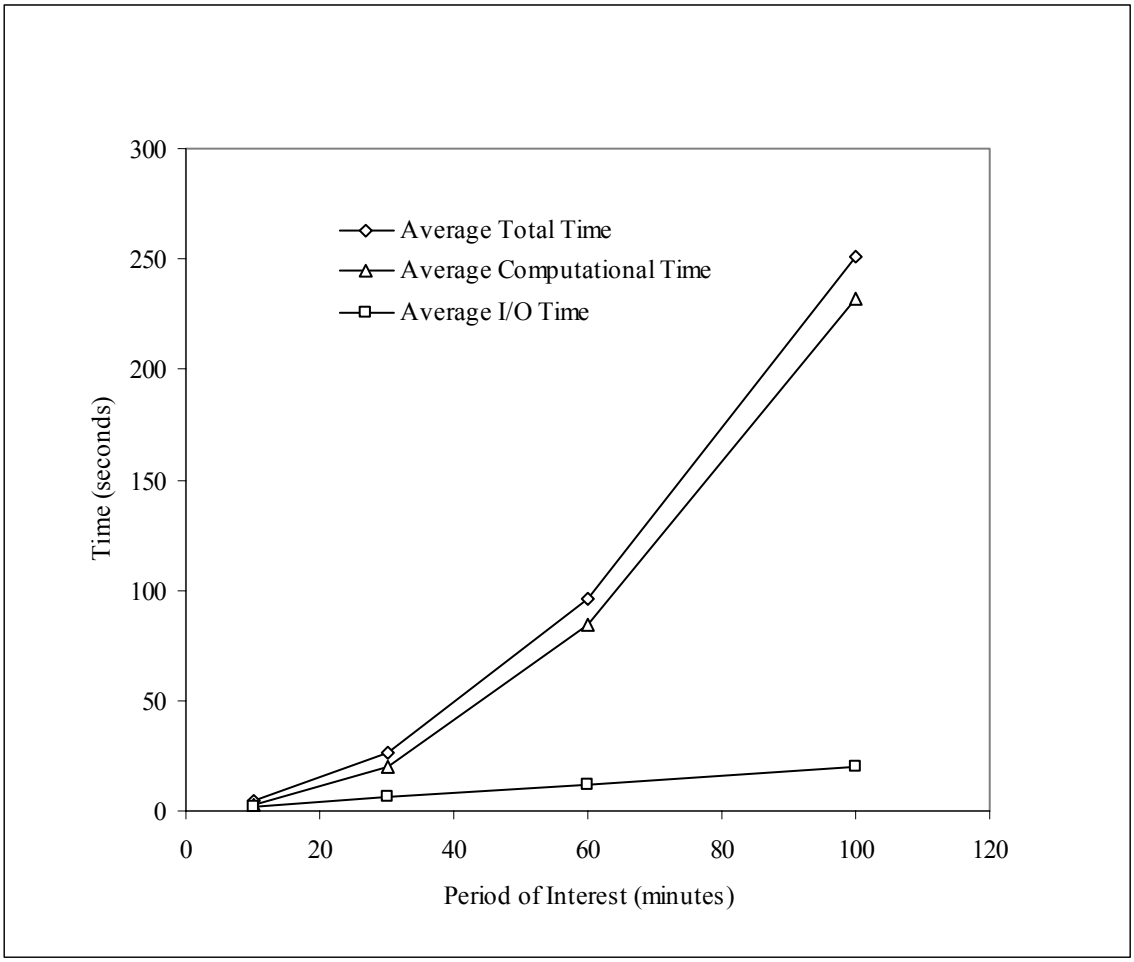


Figure 3.10 Execution Time by Parts (Parallel)

Experiment Number	Description
1	Execution of sequential SPA on machines with different architectures and operating systems
2	Execution of parallel SPA on SCORCH-IT using different numbers of processors
3	Execution of SPA on SCORCH-IT using, (i) single processor and (ii) four processors for various periods of interest

Table 3.1 Experiments Performed on SCORCH-IT

Processor	Speed	Operating System	Memory	READ*	COMPUTE*	OUTPUT*	Total*
Intel Pentium III	500 MHz	Linux	256 MB	5.85	71.99	1.02	78.86
Sun UltraSparc II	350 MHz	Unix	2 GB	8.67	152.08	1.92	162.67
Sun Ultra 2	200 MHz	Unix	512 MB	12.574	217.31	2.65	232.53

** All times are reported in seconds*

Table 3.2 Results from Experiment 1

Number of Processors	Average Total Time*	Speedup	Efficiency
1	78.86	1.00	1.00
2	44.52	1.77	0.88
3	33.27	2.37	0.79
4	26.35	2.99	0.75

** All times are reported in seconds*

Table 3.3 Speedup and Efficiency Estimates from Experiment 2

4. FAULT TOLERANCE

Chapter 4 discusses a Fourier transform based fault tolerance framework to detect data errors in real-time. In the context of real-time control of vehicular traffic networks through route guidance, faults arise due to malfunctioning detectors and/or the failure of communication links between the detector sites and the processing center. In the context of an automated on-line control architecture for real-time route guidance, the detection and correction of faults is essential for uninterrupted and effective operation of the traffic system. Existing research primarily focuses on the detection of loop detector errors. It does not address issues of the correction of erroneous data and of communication link failures. The framework also provides a capability to correct faulty data automatically in real-time. Fourier transforms are a natural choice because traffic flow data such as volume, speed, occupancy, and/or density form time sequences. While Fourier transforms have been sparsely used in the transportation arena, they have been widely and effectively used in other engineering domains. As illustrated later, the primary advantage of the proposed Fourier-based approach is that data is analyzed as is, circumventing the need for complicated models and their inherent error tendencies.

4.1 Fault Tolerance Issues for the On-line Control Architecture

Figure 4.1 illustrates the fault tolerance aspects that arise in the context of the on-line architecture for an internet-based automated remote real-time traffic system with route guidance capabilities. As shown in the figure, the data processing involves three constituent elements: (i) calibration and consistency checking, (ii) virtual system simulator, and (iii) control models. As part of the methodology to generate routing

strategies, models are used to mimic the evolution of the real system to predict future system states. However, due to the complexities inherent to the actual traffic system, such predictions may deviate from the actual field conditions. Consistency-checking models are used to bridge this gap. The calibration procedures are used to update relevant model parameter values to enable better prediction. The virtual system simulator is used to replicate the real-time traffic conditions. It has two objectives: (i) to provide fall-back support to the control models when real data is unavailable due to detector/communication link failure, and (ii) to enable the monitoring of the evolving network conditions for performance analysis. The control models are used to generate the routing to be provided to the TCC. Dynamic Traffic Assignment (DTA) models are used for this purpose.

As shown in the figure, the three major computational elements are preceded by a fault tolerance component. The fault tolerance component addresses two types of situations: (i) detector malfunction, and (ii) communication link failure. To automate the real-time operation of the on-line architecture, both these problems should be addressed within the architecture. If communication links (such as the internet) fail, straightforward and simple methods that use data stamping can be used to identify and rectify the problem. The proposed framework focuses on faults that arise due to detector malfunctions and aims to develop methodologies that automatically detect and rectify these faults.

4.2 Methodology

The fault tolerance issues addressed here focus on two types of data faults: (i) incorrect data from the field is transmitted to the processing center either due to detector

failure or malfunction, and (ii) no data is transmitted for short time durations to the processing center due to a failure in the communication link between the field and the processing center.

Several previous studies [10] have documented detector malfunctions in the context of vehicular traffic systems. While the advent of ITS has motivated the consideration of several new sensor technologies, most are characterized by malfunctions (Krogmeier et al., 1996) for at least a certain fraction of their operational time. Additionally, adverse traffic and/or weather conditions may induce temporary and/or intermittent malfunction of the sensors. Communication links to transmit field data to a remote processing center can range from satellite links to dedicated phone lines to the internet, and to a local traffic control center can range from radio towers to wireless devices to the internet. Under both scenarios, the communication links can temporarily fail either partially or catastrophically. Addressing communication link failures is outside the scope of this study since such failures can be detected efficiently using simple and direct computer communication techniques such as data stamping.

The data faults can manifest in several ways: (i) no incoming data, (ii) intermittent reception of data with gaps (with no data) in-between, (iii) erroneous incoming data, and (iv) intermittent reception of correct data with erroneous data in-between. Erroneous data can result from malfunctioning detectors (such as missing data counts and/or underestimation or overestimation of the traffic parameter) or jammed detectors (which manifests as the repetition of past data). Hence, only (iii) and (iv) are addressed here since (i) and (ii) correspond to communication link problems.

4.2.1 Fourier Transforms

This section discusses the relevant Fourier transforms theory to develop the methodologies for fault detection and data correction. It provides a basis for straightforward data analysis without complicated modeling, thereby circumventing the potential limitations of modeling assumptions.

Fourier transforms provide the ability to express any periodic function, or equivalently any function defined on a finite interval, exactly as an infinite sum of sinusoidal components [11]. If a function f(x) is defined for the interval $0 \le x \le c$, it can be represented by the Fourier series:

$$f(x) = K + \sum_{n=1}^{\infty} A_n \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{\infty} B_n \cdot \sin(\frac{2n\pi x}{c}) \qquad (4.1)$$

where K is a constant, A_n and B_n are the Fourier coefficients of the cosine and sine components respectively, and c is time interval for which a Fourier transform is to be determined. The amplitude C_n of this series is equal to $(A_n + B_n)$ and the phase $\varphi_n = \arctan\left(\frac{A_n}{B_n}\right)$. All the sinusoidal terms have angular frequencies (multiples of $2\pi/c$) so that a whole number of wavelengths can be fit into the interval $0 \le x \le c$. To enable tractability, (4.1) can be re-written as the finite sum of N sinusoidal components where a large enough N can ensure a sufficiently accurate approximation of f(x):

$$f_N(x) = K + \sum_{n=1}^{N} A_n \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{N} B_n \cdot \sin(\frac{2n\pi x}{c}) \qquad (4.2)$$

The truncated Fourier series $f_N(x)$ is easier to compute compared to the infinite one and is hence useful for applications.

A Fourier series has the following useful properties:

1) *Multiplication Property*

$$\lambda \cdot f(x) = \lambda \cdot K + \sum_{n=1}^{\infty} (\lambda \cdot A_n) \cdot cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{\infty} (\lambda \cdot B_n) \cdot sin(\frac{2n\pi x}{c}) \tag{4.3}$$

2) *Additive Property*

If $f(x) = \mathrm{K}_1 + \sum_{n=1}^{\infty} A_n^1 \cdot cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{\infty} B_n^1 \cdot sin(\frac{2n\pi x}{c})$, and

$g(x) = \mathrm{K}_2 + \sum_{n=1}^{\infty} A_n^2 \cdot cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{\infty} B_n^2 \cdot sin(\frac{2n\pi x}{c})$, then

$$f(x) + g(x) = (K_1 + K_2) + \sum_{n=1}^{\infty} (A_n^1 + A_n^2) \cdot cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{\infty} (B_n^1 + B_n^2) \cdot sin(\frac{2n\pi x}{c}) \tag{4.4}$$

The same properties also hold for the truncated series:

$$1)\ \lambda \cdot f_N(x) = \lambda \cdot K + \sum_{n=1}^{N} (\lambda \cdot A_n) \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{N} (\lambda \cdot B_n) \cdot \sin(\frac{2n\pi x}{c}) \tag{4.5}$$

2) If $f_N(x) = \mathrm{K}_1 + \sum_{n=1}^{N} A_n^1 \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{N} B_n^1 \cdot \sin(\frac{2n\pi x}{c})$, and

$g_N(x) = \mathrm{K}_2 + \sum_{n=1}^{N} A_n^2 \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{N} B_n^2 \cdot \sin(\frac{2n\pi x}{c})$, then

$$f_N(x) + g_N(x) = \mathrm{K}_1 + \mathrm{K}_2 + \sum_{n=1}^{N} (A_n^1 + A_n^2) \cdot \cos(\frac{2n\pi x}{c}) + \sum_{n=1}^{N} (B_n^1 + B_n^2) \cdot \sin(\frac{2n\pi x}{c}) \tag{4.6}$$

The Fourier series (4.1) can also be formulated using complex algebra. Using the definition $e^{i\theta} = \cos\theta + i \cdot \sin\theta$, where the first term on the RHS corresponds to the real part and the second term corresponds to the imaginary part, the Fourier series can be written as follows:

$$f(x) = Z_0 + \sum_{n=1}^{\infty} (Z_n \cdot e^{2\pi i \cdot n \cdot x/c} + Z_{-n} \cdot e^{2\pi i \cdot (-n) \cdot x/c}) \tag{4.7}$$

where, $Z_n = (1/c)\int_0^c [e^{-2\pi i \cdot n \cdot x / c} \cdot f(x)]dx$. (4.8)

For n=0, $Z_0 = (1/c)\int_0^c f(x)dx = K$. (4.9)

From (8) and (9), we can re-write (7) as:

$$f(x) = \sum_{n=-\infty}^{\infty} Z_n \cdot e^{2\pi i \cdot n \cdot x / c} \quad (4.10)$$

The amplitude C_n is equal to $2 \cdot |Z_n|$ and the phase is $arg(Z_n)$. The properties mentioned for the continuous Fourier transforms also hold for the discrete case. In the study, the following complex algebra based expression of the Fourier series is used:

$$f(x) = K + \sum_{n=1}^{N} \mathrm{Re}(n) \cdot \cos[2\pi \cdot \nu(n) \cdot x] + \mathrm{Im}(n) \cdot \sin[2\pi \cdot \nu(n) \cdot x] \quad (4.11)$$

where, *Re*(n) and *Im*(n) are, respectively, the real and imaginary Fourier coefficients, and ν(n) is the frequency. The frequency is equal to $\nu(n) = \frac{n-1}{N \cdot \iota}$, where ι is the time interval between two consecutive data points (in seconds), labeled *data acquisition interval.* The amplitude *ampl*(n) is equal to $\sqrt{Re^2(n) + Im^2(n)}$ and the phase is $arg\left(\frac{Re(n)}{Im(n)}\right)$.

The Fourier spectrum is a representation of the real and imaginary coefficients with respect to their corresponding frequencies. The following example illustrates a simple sinusoidal function and its Fourier spectrum.

Example

Consider the function f(t) = 10·sin(10·2πt) + 5·sin(20·2πt). It consists of two functions: $f_1(t)$ = 10·sin(10·2πt) and $f_2(t)$ = 5·sin(20·2πt). Figure 4.2 illustrates f(t) in terms of two sinusoidal functions $f_1(t)$ and $f_2(t)$. The Fourier spectrum of the time series

f(t) is shown in Figure 4.3. It indicates that the original time series consists of the functions $f_1(t)$ and $f_2(t)$. The function $f_1(t)$ is represented by the amplitude 10 at a frequency 10 Hz, and $f_2(t)$ is represented by the peak at frequency 20 Hz with an amplitude 5.

4.2.2 Overview of Methodology

The Fourier theory discussed in Section 4.2.1 is used to analyze real-time streaming traffic data from field detectors to identify data faults. The commonly available traffic flow measures from detectors include volume V(t), speed S(t) and occupancy O(t), as a function of time t. Fourier transforms are used to identify the underlying characteristics of this data for inferring on any data abnormalities that occasionally arise.

In this section, the main steps of the methodology are briefly discussed. The methodology has the following steps:

1) Initial Data Processing

The historical data set is first smoothed using an exponential smoothing technique. Then, time sequences of N data points are constructed (discussed in detail in Section 4.2.3).

2) Training

The smoothed data sets from the initial processing procedure are used to determine the necessary threshold parameters for the Fourier-based methodology (discussed in Section 4.2.4).

3) Detection of Data Faults

Streaming data is analyzed in order to detect data faults. Data smoothing is performed for newly arriving data (discussed in Section 4.2.5).

5) Data Correction

A Fourier-based heuristic is used to correct the erroneous data. The heuristic isolates the erroneous part of the data and predicts their likely correct values (discussed in Section 4.2.6).

4.2.3 Initial Data Processing

The historical abnormality-free data for a road cross-section is first smoothed to reduce the inherent volatility of real traffic data. Figure 4.4 shows the raw and smoothed data for a link on the Athens network. The raw data is characterized by temporary traffic flow fluctuations that represent noise in the acquired data. The smoothing is performed using an exponential smoothing technique. Suppose d(t) is a data point for time t. The smoothed data value $\bar{d}(t)$ of this data point is obtained using the P preceding data points. The number of preceding data points used in the smoothing procedure is dependent on the desired smoothing level. The smoothing increases with increasing P. The exponential smoothing technique is described by the following equation:

$$\bar{d}(t) = \frac{\sum_{i=1}^{P} e^{-\alpha i} \cdot d(t-i+1)}{\sum_{i=1}^{P} e^{-\alpha i}} \tag{4.12}$$

where α is a smoothing parameter. Reducing α increases smoothing as it implies an increased contribution from past data points.

The smoothed historical abnormality-free data for time $t = t_0, \ldots. t_e$, where t_0 is the first data point and t_e is the last data point of the historical data set, is grouped into N-point time sequences. The first sequence starts at t_0 and ends at $t_0 + (N-1)$. The next time sequence is obtained by shifting the current time sequence by one (starts at $t_0 + 1$ and

ends at t_0 + N). Each data set contains N data points which is also equal to the number of Fourier points. The number of Fourier points is set equal to N so that the transformation of the time sequence to its Fourier series has the minimum approximation error.

4.2.4 Training

The first step of the training is performed by computing the Fourier transform and the Total Fourier Area (TFA) for each N-point time sequence of the historical abnormality-free data set. For each time sequence the TFA is defined as the sum of all Fourier coefficients:

$$TFA(t) = \sum_{n=1}^{N} \mathrm{Re}(n) + \mathrm{Im}(n) \tag{4.13}$$

After computing the Fourier transform and TFA for all time sequences in the training set, the relative difference of successive TFAs, called the DTFA, is computed for each t:

$$DTFA(t) = 100 \cdot \left[\frac{TFA(t) - TFA(t-1)}{TFA(t-1)} \right] \tag{4.14}$$

The DTFA(t) is an indicator of the magnitude of change in traffic conditions from interval t-1 to t. The DTFAs are used to determine the maximum absolute DTFA for abnormality-free conditions. It is used in the next section to detect data abnormalities. Figure 4..5 shows the DTFAs for one day on a link on the Athens, Greece, network for ι = 90 seconds and N = 256.

The minimum and maximum real and imaginary Fourier coefficients over all N-point time sequences of the historical data set are used to define the Feasible Fourier Zone (FFZ) for abnormality-free data. The FFZ represents the region in which the coefficients of the Fourier transforms of an abnormality-free time sequence reside with

high likelihood. It implies that if the Fourier coefficients of a time sequence lie outside the zone, the associated data is likely to possess an abnormality. Hence, the minimum and maximum coefficient values are used to determine the boundaries of the FFZ as shown in Figure 4.6. Here, the FFZ of the imaginary coefficients for one day of abnormality-free data from a road cross-section on the Athens network is depicted. The FFZ will be used by the data correction heuristic in Section 4.2.6.

4.2.5 Detection of Data Faults

The DTFAs computed for real-time streaming data are used to identify a data abnormality. The Fourier transform for the sequence of N data points at time t is computed. If $|DTFA(t)| > \lambda$, where λ is a threshold value, a data abnormality is detected by the Fourier-based methodology. λ is the maximum absolute DTFA observed in the training phase discussed in Section 4.2.4. This approach can adapt to demand changes over a period of time through a retraining step. During the retraining, a new value of λ that reflects the changed demand conditions is computed using recent data and becomes the basis for the abnormality detection criterion. Four criteria identify data faults:

Criterion 1: Volume, speed, and occupancy cannot be higher than their maximum possible values in a traffic system. If this criterion is violated, the abnormality is identified as a data fault.

Criterion 2: If volume (V), occupancy (O), and speed (S) in successive intervals are equal, the abnormality is identified as a data fault. This criterion implies a jammed detector that leads to the repetition of past data. It can be checked even before computing the DTFAs.

Criterion 3: Speed and density (D) cannot be simultaneously equal to zero. If $S \rightarrow 0$, then $D \neq 0$ as D tends to the jam density. If $D \rightarrow 0$, then $S \neq 0$ as it tends to the free flow speed. If this criterion is not satisfied, the abnormality is classified as a data fault. If it is satisfied, a possible incident is indicated.

Criterion 4: The Fault Identification Ratio (FIR) cannot be significantly different from 1:

$$FIR = \frac{V}{S \cdot D} \tag{4.15}$$

This is due to the traffic flow identity $V = S \times D$. Hence, if the FIR is significantly different from 1, the abnormality is classified as a data fault. While ideally the traffic flow identity should be conserved, in reality the FIR may differ from 1 due to the mechanism to measure density based on occupancy. Density is inferred from the occupancy as follows:

$$D = \frac{O}{100 \cdot (L_v + L_l)} \tag{4.16}$$

where O is the occupancy in percent, L_v is the typical vehicle length, and L_l the loop length. While L_l is a constant, vehicles have varying lengths requiring L_v to be an average measure, introducing some approximation in inferring D. Hence, the FIR may not be exactly equal to 1 for a traffic system. Fault-free conditions are likely to exist when the FIR = $1 \pm \varepsilon$, where ε is a threshold value whose magnitude, while small, depends on the particular traffic network being analyzed (based on the vehicle mix). When this criterion is violated, the abnormality is classified as a data fault.

4.2.6 Correction of Data Faults

When a data abnormality is identified as a fault, a data correction heuristic is activated to correct the erroneous data. The primary objective of this heuristic is to

predict the likely correct data. Consequently, it could also be used to predict data values for the near-term or medium-term future. Such a capability is significant when detectors malfunction in a traffic system with automated operational control.

The procedure begins with the last data point added to a time sequence being identified as an abnormality due to a data fault. The resulting N-point time sequence with the last data point deliberately set equal to zero is called a *faulty time sequence*. Figure 4.7 illustrates such a time sequence for N = 256. The faulty time sequence is updated using the "correct" data point predicted by the heuristic. This is done by using a *fault time sequence*, which has the first (N-1) points equal to zero and the last data point predicted by the heuristic. The faulty time sequence is added to the fault time sequence to obtain the corrected time sequence. Figure 4.8 illustrates a fault time sequence for N = 256. As discussed in Section 4.2.1, the addition of two transforms is equivalent to the transform of the addition of the two sequences (see 4.6). Hence, if the Fourier transforms of the faulty and fault time sequences are added and reversed to a time sequence, the resulting time sequence will have its first N - 1 data points equal to the original ones and the last one (N^{th}) equal to the predicted value. This is the mechanism by which the corrected time sequence is obtained.

A key advantage of the proposed approach is that the effect of adding just one data point to generate the corrected time sequence influences the coefficients of all frequencies. Thereby, if the proposed data point is incorrect, then its Fourier coefficients lie outside the FFZ. This provides a simple and direct interpretation for the proposed Fourier-based heuristic.

4.3 Fault Detection Experiments

4.3.1 Description of the Experiments

The fault detection methodology was analyzed by deliberately introducing data faults to the volume, occupancy, and speed time sequence every 4 minutes. Introducing faults independently and periodically ensured that a fault could not be correlated or amplified by a previous one. Twenty data fault scenarios were explored as shown in Table 4.1. Each scenario was tested on a minute-by-minute basis for the 24 days to replicate continuous monitoring. Hence, each fault scenario had 15,240 experiments out of which 3,792 were faulty conditions.

The fault scenarios described in Table 4.1 represent different levels of fault likelihood and severity. There are 7 scenarios each for fault and speed, and six for occupancy. The most severe and likely are scenarios are 1, 2, 3, 8, 9, 14 and 15. Scenario 1 represents the case of a jammed detector that repeats the same data over and over again. The others (2, 3, 8, 9, 14, and 15) represent malfunctioning detectors from which erroneous data of small magnitude is received. These scenarios are the most likely ones based on an analysis of field data from Berkeley, Athens (Greece), and West Lafayette.

The other scenarios (4-7, 10-13, and 16-20) are more conservative and are primarily explored to investigate the limits of the methodology. Most of these scenarios relate primarily to detector hardware imprecision that generate faulty and/or systematically biased data, requiring hardware fault diagnosis and repair. Examples of these scenarios are 11-13 and 17-20. Most detector accuracies range from 85 to 90%. Faults that were not detected when they occurred were not detected at all. Another issue to be noted is that sometimes the methodology recognizes a fault, but is unable to specify

whether it lies with volume, occupancy, or speed data. The methodology identifies such faults as "fault-in-all" time sequences. Scenario 1 is a good example of this possibility. The implications of identifying faults in all (volume, speed, occupancy) sequences is that the associated data correction heuristic attempts to correct all of them raising issues of computational efficiency in addition to small losses in accuracy. However, these issues are insignificant.

4.3.2 Discussion of Results

The detection rate (DR) for the most likely fault scenarios (1, 2, 3,8, 9, 14, and 15) was found to be 100% in all cases, and there were no false alarms. Due to the 100% DR rates, no figures have been generated for these scenarios. However, the results are presented for the less realistic cases. They also have a 0% false alarm rate (FAR).

Figure 4.9 illustrates the DR for scenario 4. It has a value 97.60% for faults in volume, and 98.23% when faults classified as fault-in-all are also included (this is denoted by the black fault-in-all symbol in the figure). This scenario represents the case that due to some detector malfunction, volume has significantly lower values (0 to 500 vph, randomly allotted) than under the actual conditions (800 to 1,500 vph) that are observed over 24 days of the Berkeley data. The DR for scenario 5 is lower than that of scenario 4, as illustrated in Figure 4.10. It averages 84.49% and 85.47% under fault-in-all. This is because the 0 to 800 vph range of scenario 5 is closer to the actual field data compared to the 0 to 500 vph range of scenario 4. Hence, scenario 4 exhibits better DR. It should be noted that both these scenarios are not likely in practice.

Scenarios 6 and 7 are unlikely in practice, and point to a hardware problem because of the systematic bias rather than a data fault issue. As illustrated in Figure 4.11

scenario 6 has DRs of 75.42%, and 78.32% (fault-in-all). It is difficult to detect data faults which are systematically underestimated. It also led to more faults being classified as fault-in-all. Scenario 7 has DRs of 81.33% and 95.26% (fault-in-all), as illustrated in Figure 4.12. Here, the volume is systematically overestimated by 50%, and for this reason more faults are likely to be detected but classified under fault-in-all. This implies that the methodology can detect overestimated volume better than underestimated volume. If the volume is overestimated or underestimated by less than 50%, the methodology is not effective as the DR drops dramatically to less than 50%. Hence, 50% underestimation or overestimation represent the limit of the effectiveness of the approach. The methodology can be made more sensitive, but would introduce trade-offs in terms of increased FARs. Scenario 10 represents detector malfunction vis-à-vis occupancy whereby it generates higher occupancies than normal (in the range 15 to 30% randomly). The average DR values are 86.58% and 88.13% (fault-in-all) as illustrated in Figure 4.13. As per the Berkeley data, occupancy rarely goes beyond 15% for the 24-day period. Scenario 11 (Figure 4.14) represents another case which is likely to occur due to hardware problems rather than data faults as illustrated by the systematic underestimation of occupancy by 30%. Assuming a data fault, the average DR is only 4.35%, and the fault-in-all class has DR 69.07%. The methodology incorrectly interprets that only a small fraction of the faults identified are due to occupancy data faults. However, if occupancy is underestimated by a higher margin, as in scenario 12, the methodology is better able to detect the faults, though it also points to a hardware problem. Thereby, almost all faults are detected, 98.71% (fault-in-all), though those correctly attributed to occupancy average 49.82% (see Figure 4.15). As before, when the quantity is

overestimated, the ability to attribute them to occupancy data faults is much better. As illustrated in Figure 4.16, the average is 80.30%, and for fault-in-all is 81.62%, for scenario 13. Further analysis suggests that occupancy systematically overestimated or underestimated by less than 30% cannot be effectively detected by the methodology as the DR drops to less than 40%. Here, almost all faults are classified as fault-in-all. Hence, 30% overestimation or underestimation is limit of effectiveness of the methodology vis-à-vis occupancy. While the methodology can be made more sensitive, trade-offs with FAR arise as discussed for volume.

Scenario 16 represents a relatively realistic case of detector malfunction whereby speeds take significantly lower values (in a range of 0 to 30 mph randomly) than the actual ones. As per the Berkeley data, speed dropped to less than 30 mph only under very severe incidents over the 24-day period. As illustrated in Figure 4.17, the associated DRs are high, 98.95% and 99.34% (fault-in-all). The DR for scenario 17 is significantly lower than for scenario 16, averaging 74.87% and 76.16% (fault-in-all), as shown in Figure 4.18. The systematic underestimation of speed by 25% is more representative of hardware problems. It is interesting to note that even in this case a high percentage of faults are detected and correctly classified.

Scenario 18 is also primarily a case of hardware problems rather than data faults. However, since speed is underestimated by a larger value than in the previous case, it is more likely to occur in practice. Figure 4.19 illustrates the higher DR of 95.81% and 96.78% (fault-in-all) for Scenario 18. The DR for scenario 19 is relatively lower, averaging 77.08% and 77.27% (fault-in-all), as depicted in Figure 4.20. The methodology performs worse for a 30% overestimation compared to a 30% underestimation. This

again highlights the strength of the methodology as 30% speed underestimation is much more likely than a 30% overestimation. When speed is overestimated by 60% (scenario 20), the methodology performs robustly as illustrated by the DR of 90.98% and 99.17% (fault-in-all) in Figure 4.21.

Based on further experiments not discussed here, overestimation by less than 25% or underestimation by less than 30% cannot be robustly detected by the methodology as the DR drops to less than 50%. Also, the fraction of faults classified as "fault-in-all" increases dramatically. These bounds provide the limits for the methodology vis-à-vis speed detection. Greater sensitivity can be generated, but with trade-offs in terms of FAR.

4.4 Insights

The performance of the Fourier-based methodology for fault detection is robust for the most likely scenarios, with DR of 100% and FAR of 0%. However, other scenarios were analyzed to derive insights on the limits of the methodology. For less severe and less likely faults, the DR averages 76.14% and 87.14% (fault-in-all). The methodology is more sensitive to speed-related faults but less sensitive to occupancy-related faults, and even less to volume-related ones. The time-to-detection (TTD) was always 0 seconds implying that a fault which is detected, is identified at the time it is introduced.

Fourier-based methodologies are used for fault detection and correction. Fourier transforms analyze data directly without need for predictive and/or complicated modeling, thereby circumventing likely modeling errors. The proposed Fourier-based methodologies require little training, and use little data for training. Hence, the training is

simple and straightforward. Thereby, the Fourier-related parameters used in the methodology are simple to compute. However, the parameters need to be re-computed if significant demand/supply changes occur over time. Also, the parameters are not transferable to new sites, but can be computed easily with the availability of data. Another issue vis-à-vis data is the requirement of abnormality-free data for training purposes.

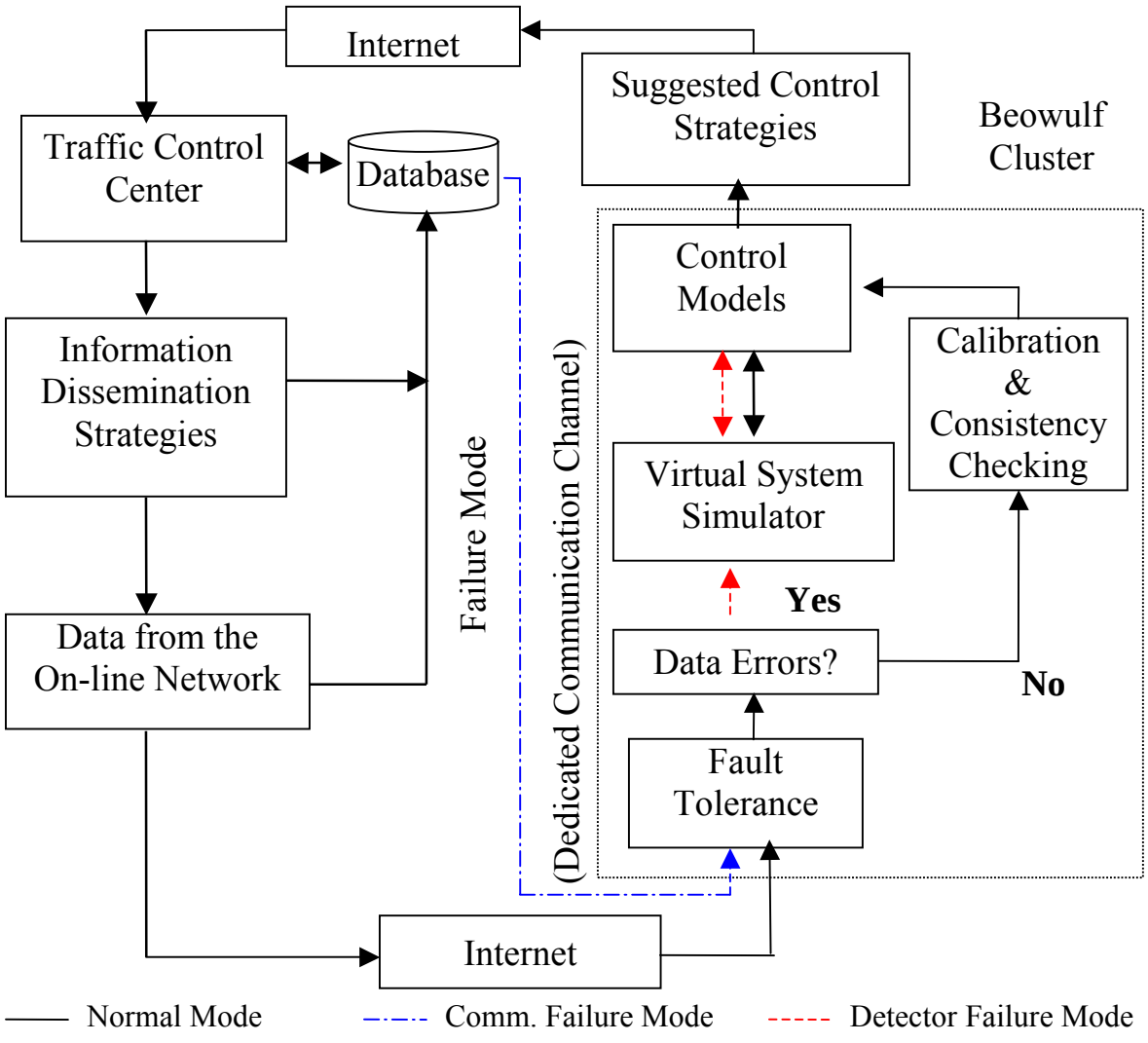


Figure 4.1 Fault tolerance aspects of the on-line control architecture for real-time route guidance

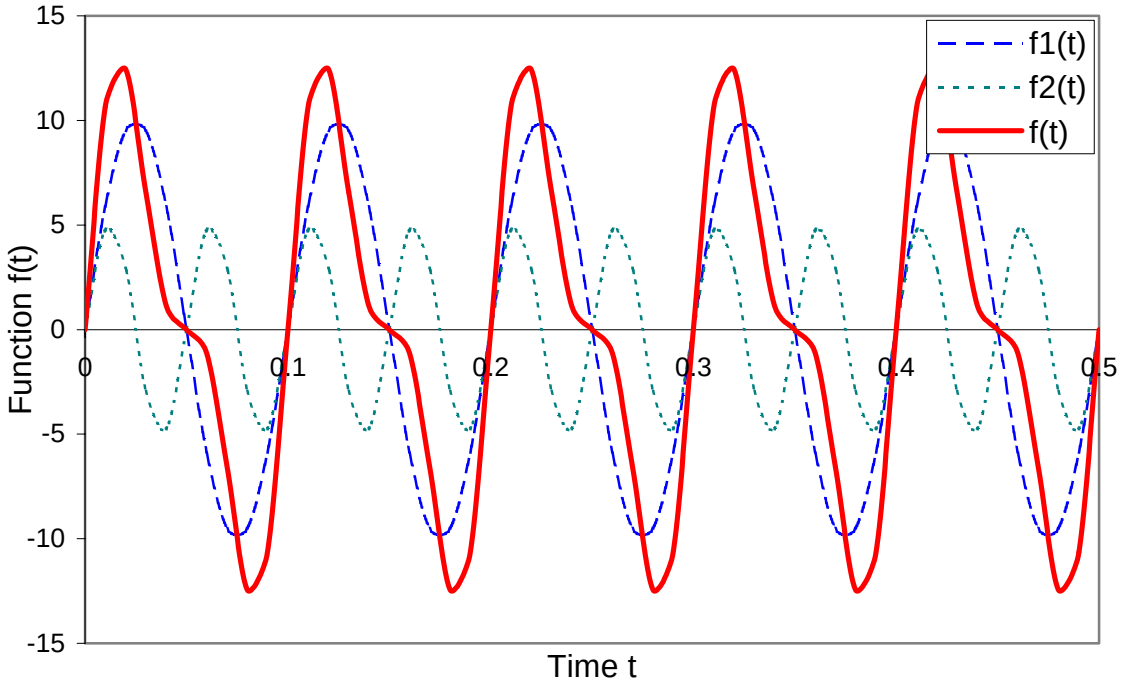


Figure 4.2 Function f(t) illustrated as consisting of two sinusoidal functions $f_1(t)$ and $f_2(t)$

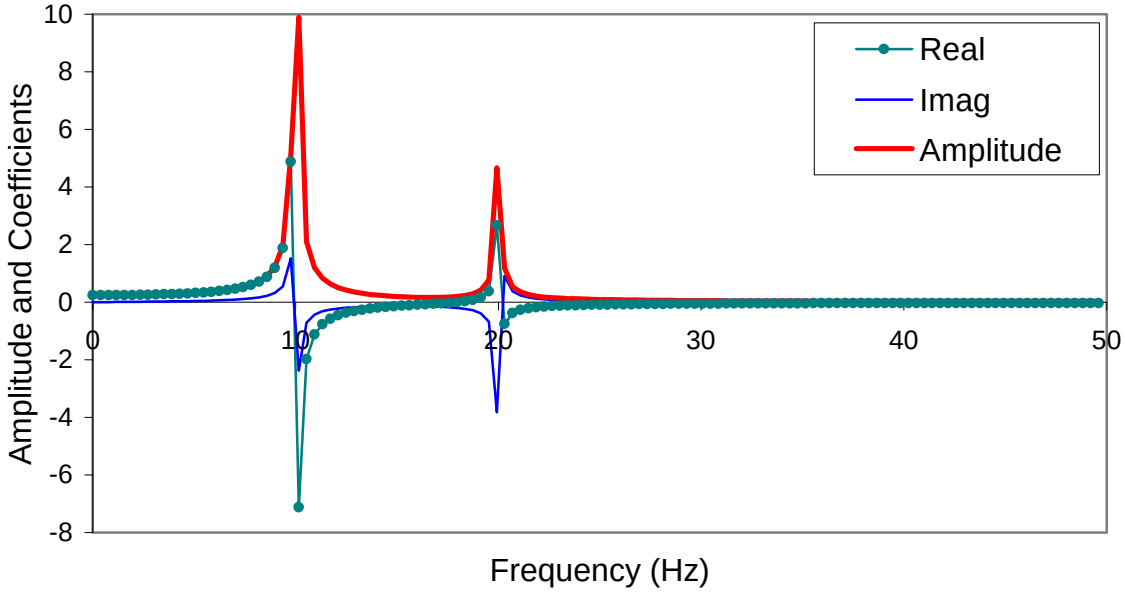


Figure 4.3 The Fourier spectrum of f(t)

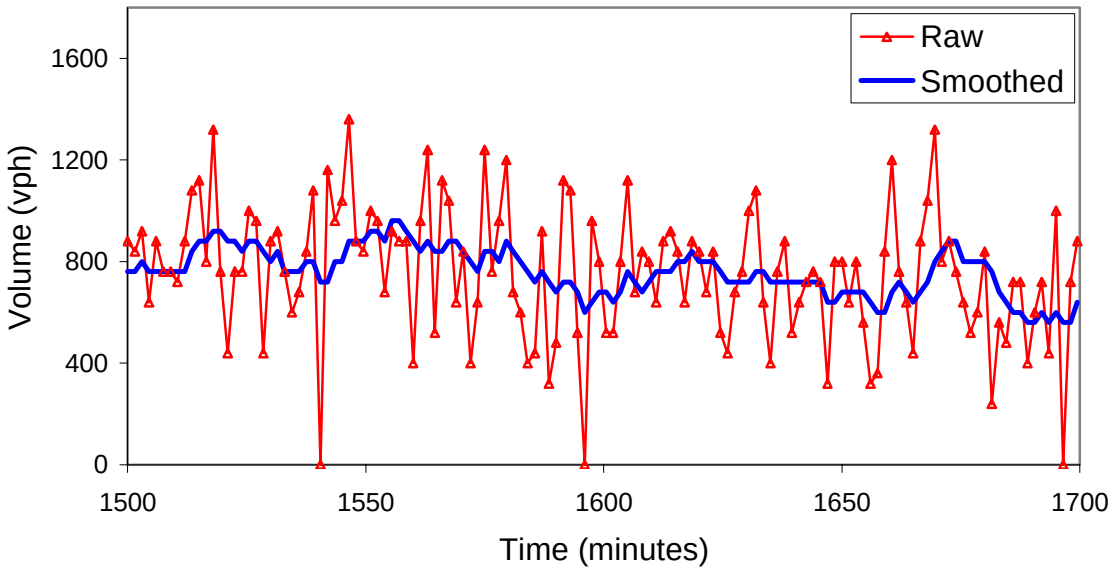


Figure 4.4 Raw and smoothed volume data for a link on the Athens network

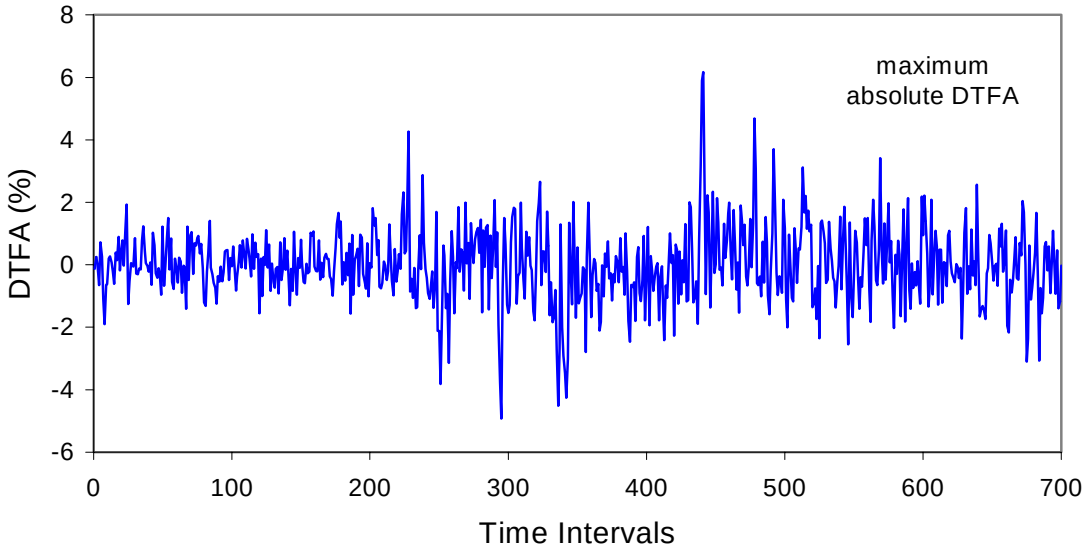


Figure 4.5 DFTAs for one day on a link on the Athens network

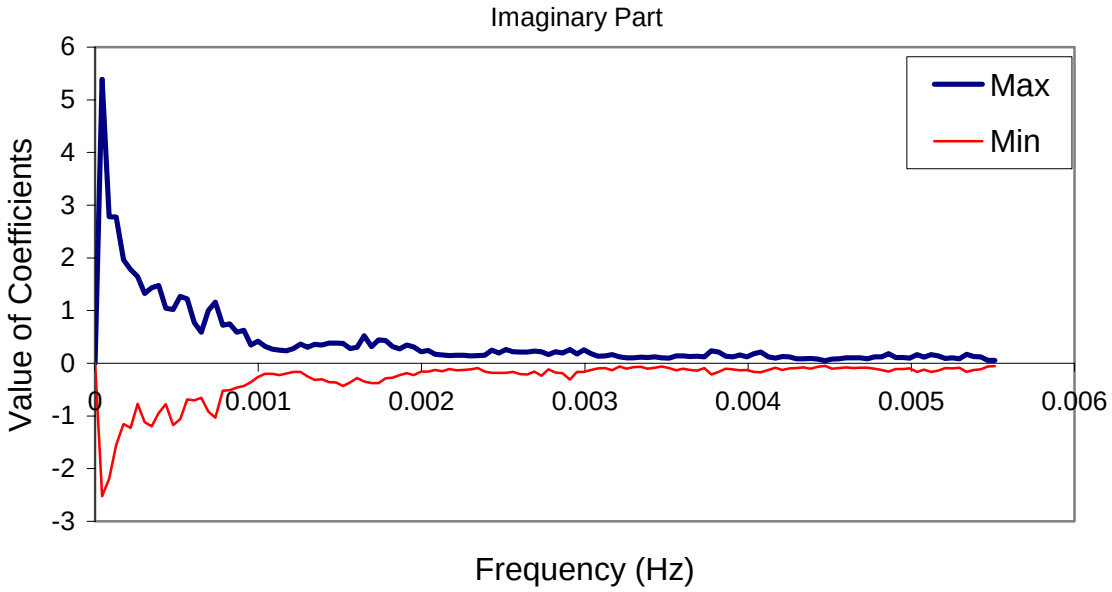


Figure 4.6 FFZ of the imaginary coefficients for one-day data from the Athens network

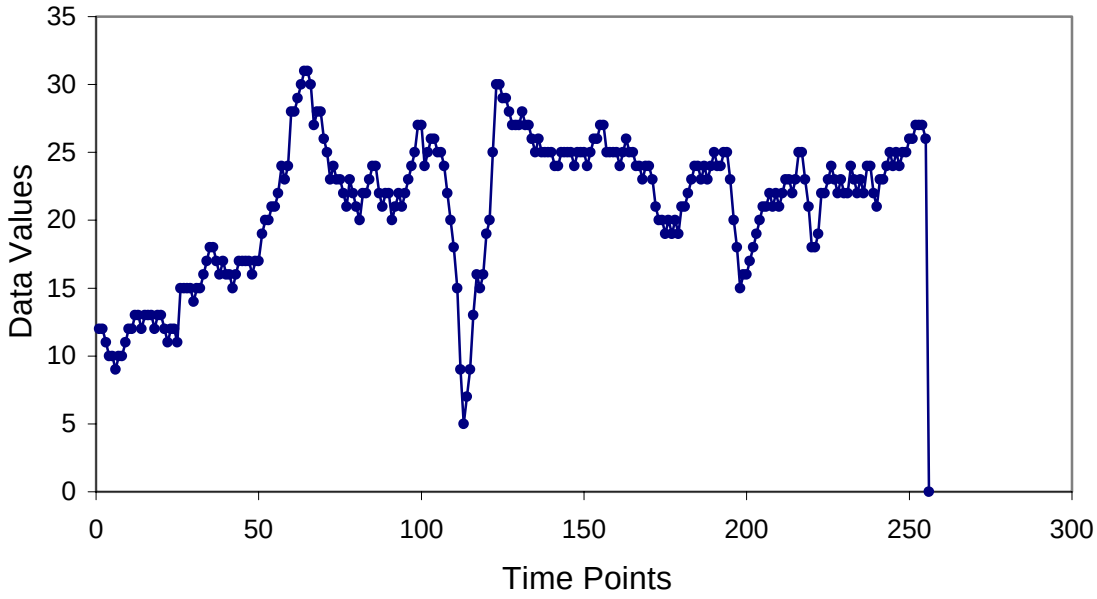


Figure 4.7 A faulty time sequence for N = 256

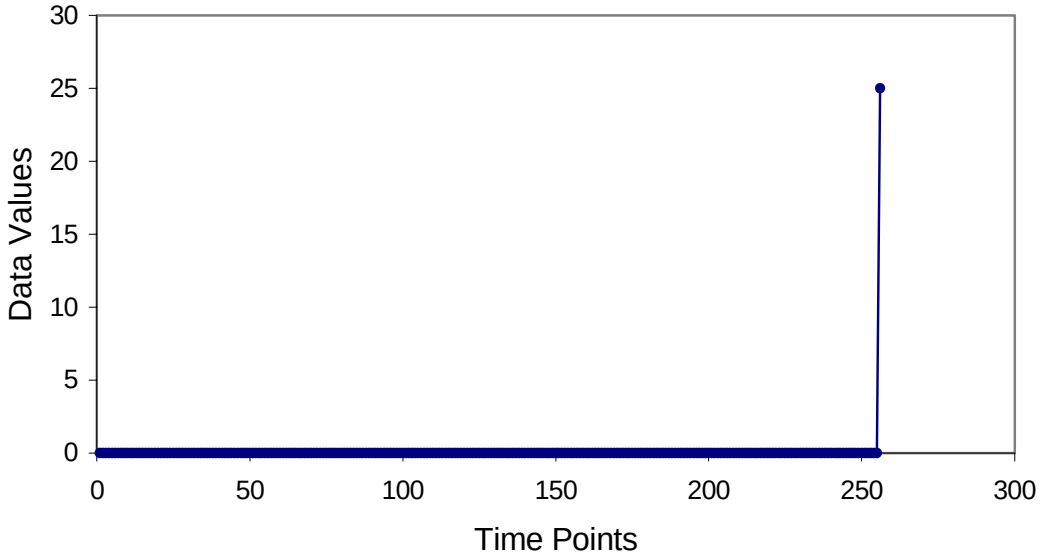


Figure 4.8 A fault time sequence for N = 256

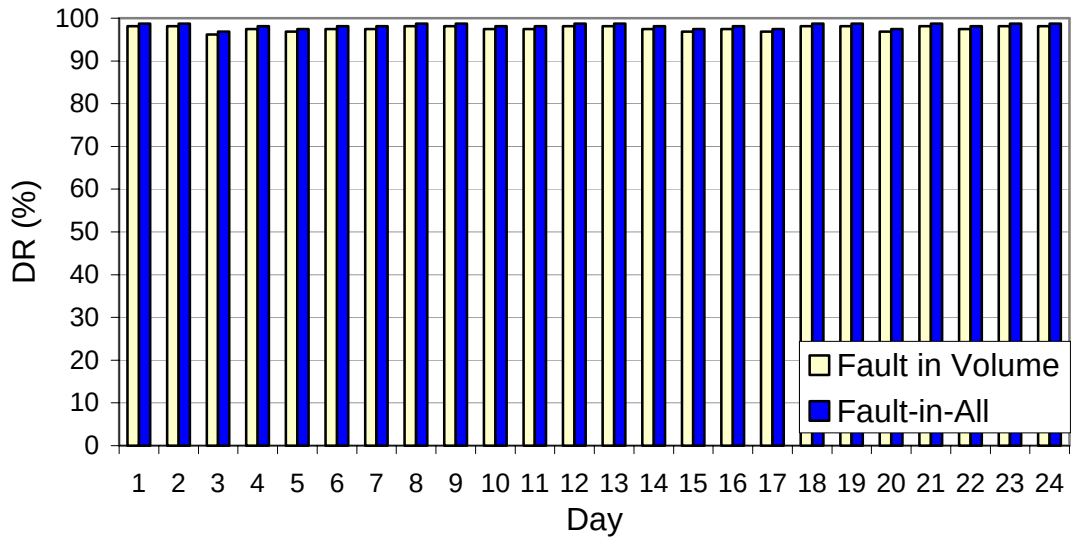


Figure 4.9 DR for scenario 4 (volume = 0 to 500 vph randomly)

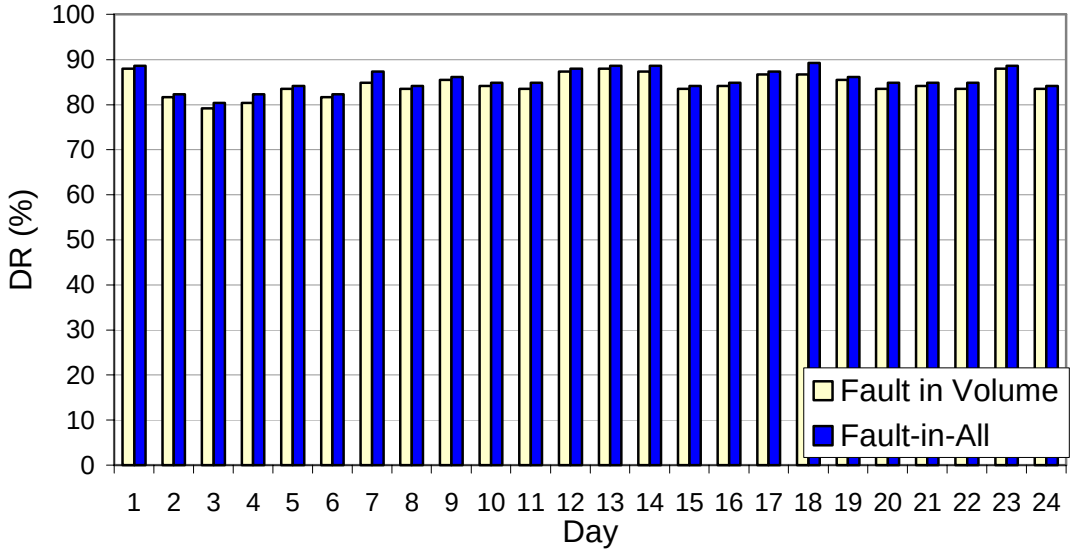


Figure 4.10 DR for scenario 5 (volume = 0 to 800 vph randomly)

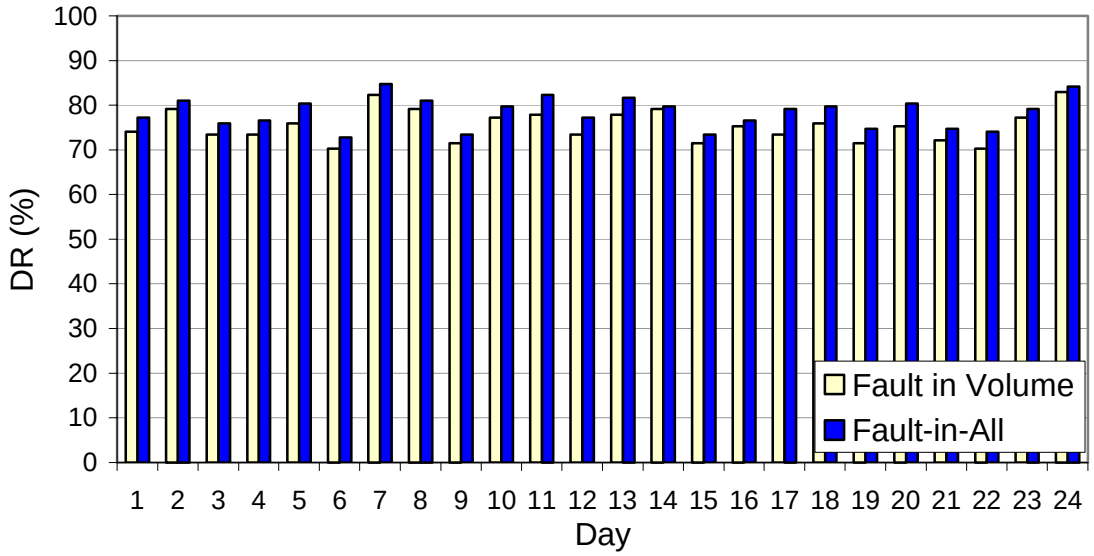


Figure 4.11 DR for scenario 6 (volume = 50% underestimated)

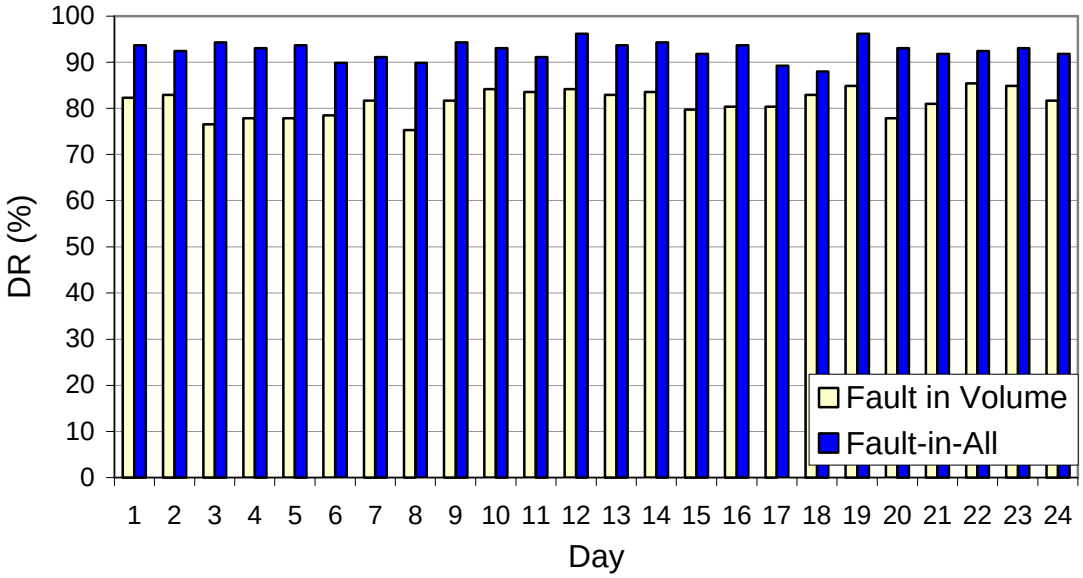


Figure 4.12 DR for scenario 7 (volume = 50% overestimated)

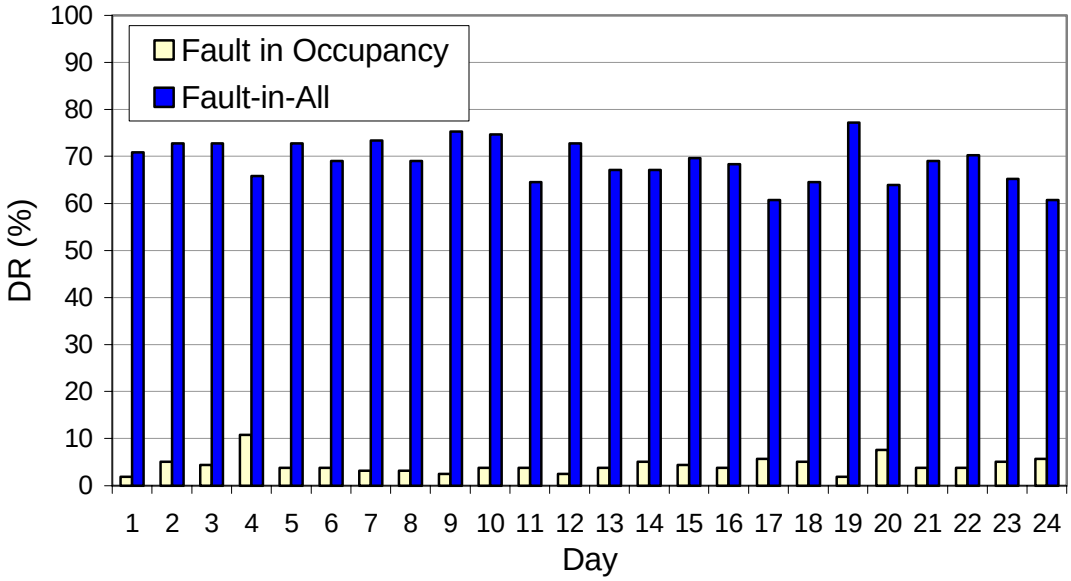


Figure 4.13 DR for scenario 10 (occupancy = 15 to 30% randomly)

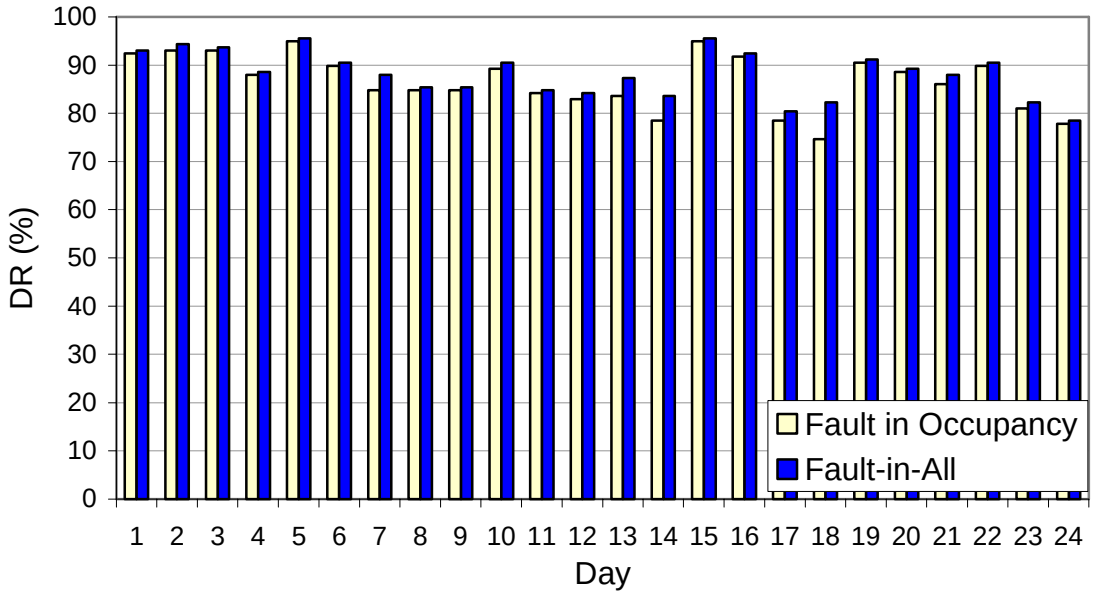


Figure 4.14 DR for scenario 11 (occupancy = 30% underestimated)

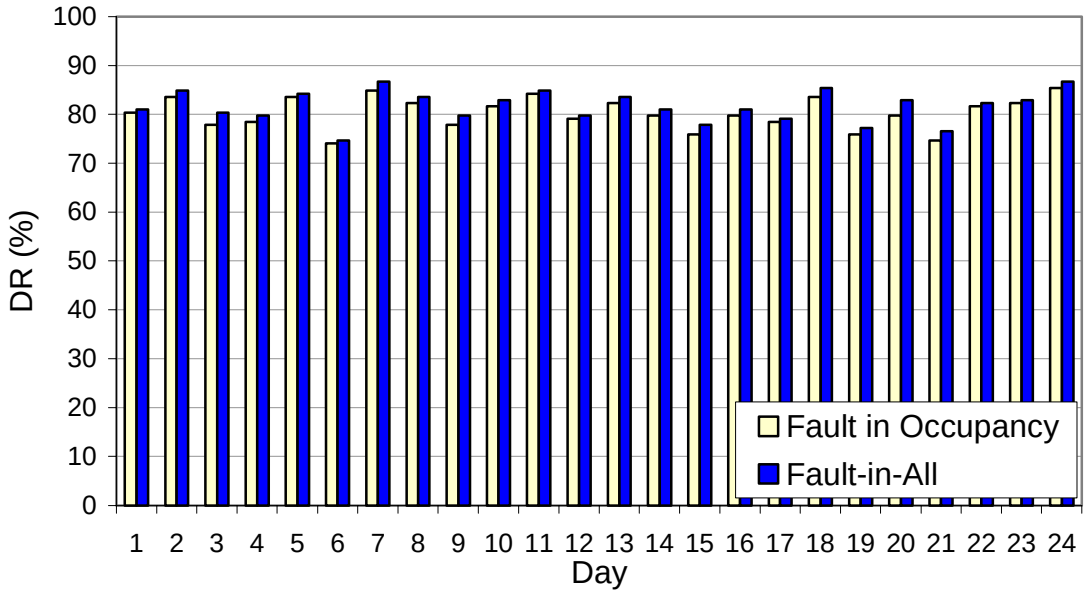


Figure 4.15 DR for scenario 12 (occupancy = 40% underestimated)

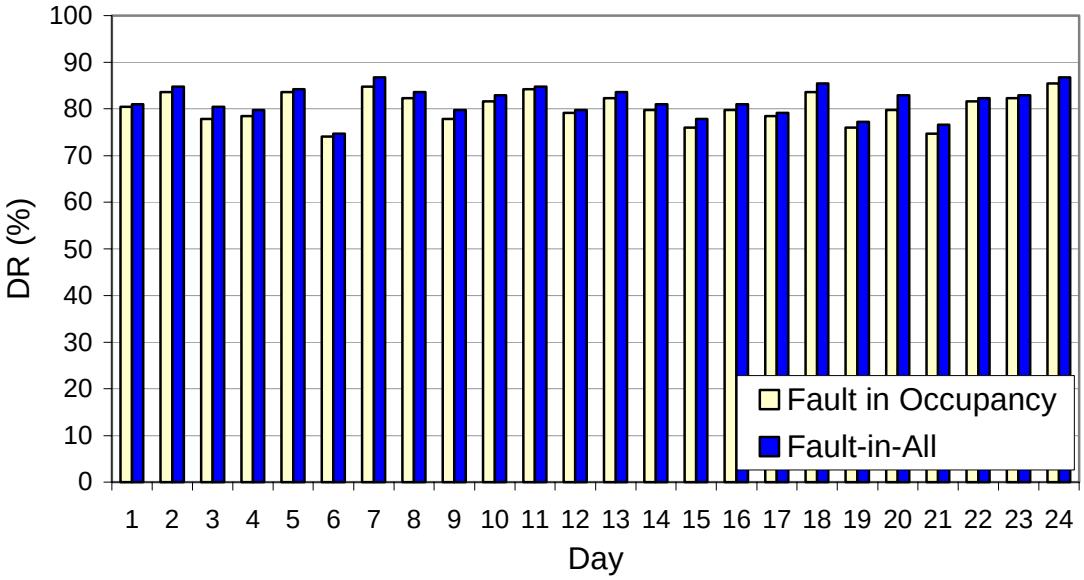


Figure 4.16 DR for scenario 13 (occupancy = 40% overestimated)

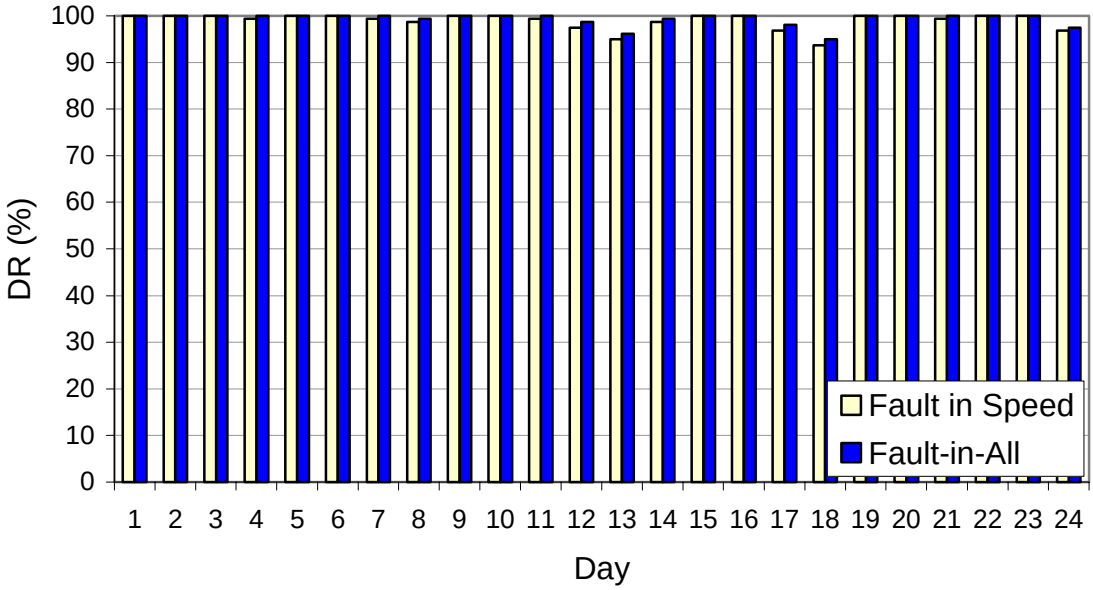


Figure 4.17 DR for scenario 16 (speed = 0 to 30 mph randomly)

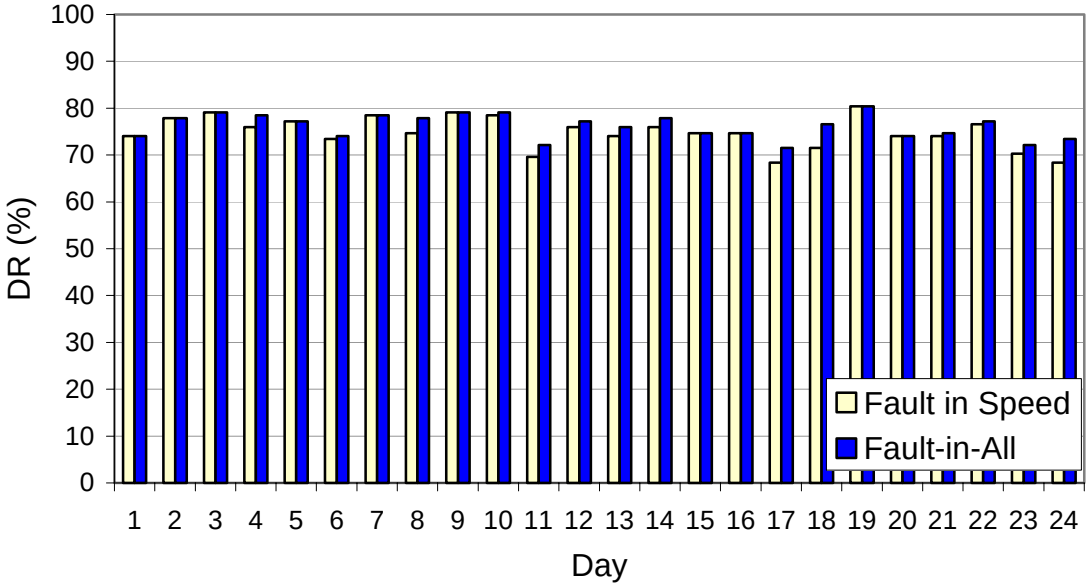


Figure 4.18 DR for scenario 17 (speed = 25% underestimated)

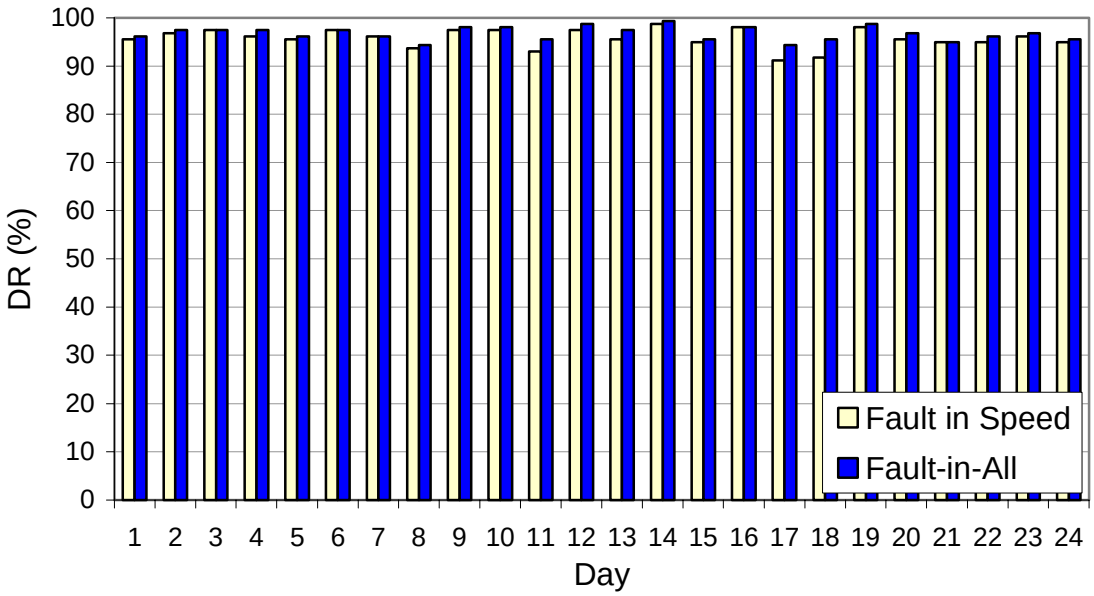


Figure 4.19 DR for scenario 18 (speed = 30% underestimated)

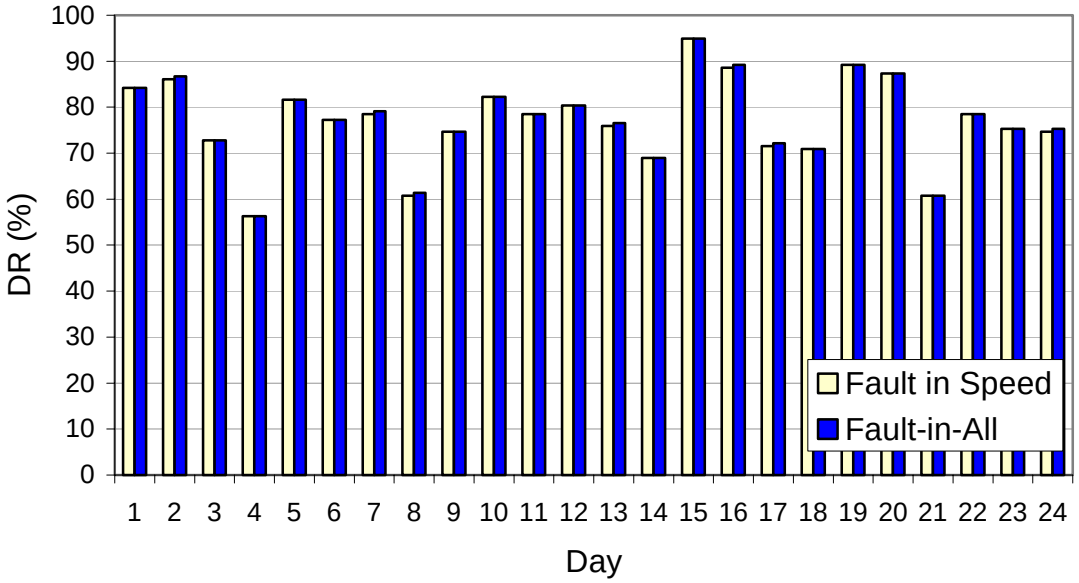


Figure 4.20 DR for scenario 19 (speed = 30% overestimated)

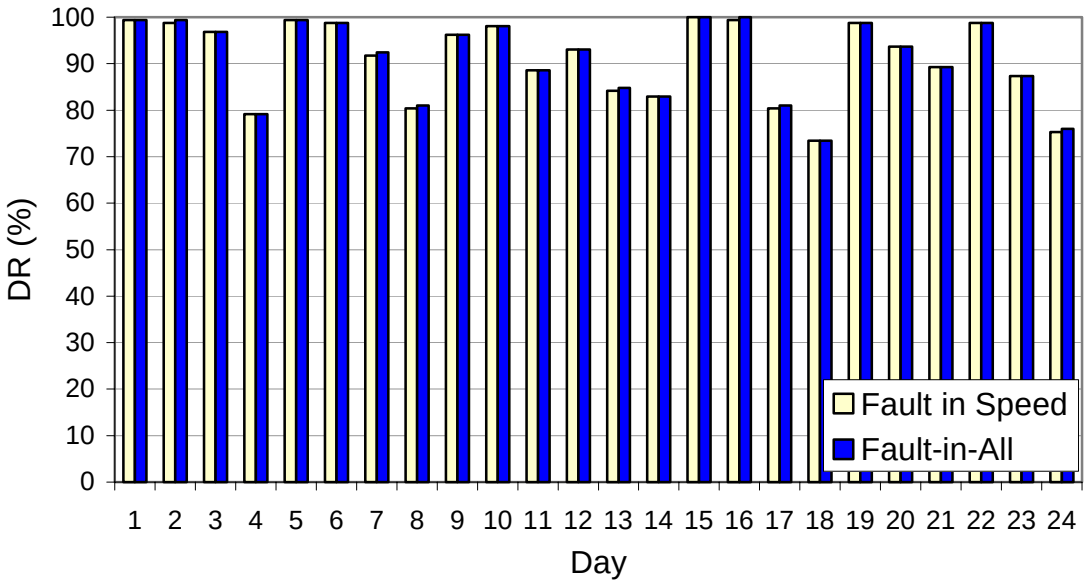


Figure 4.21 DR for scenario 20 (speed = 60% overestimated)

Scenario Number	Volume (vph)				Occupancy (%)				Speed (mph)			
	Specific value	Randomly in a Range	Under-estimated	Over-estimated	Specific value	Randomly in a Range	Under-estimated	Over-estimated	Specific value	Randomly in a Range	Under-estimated	Over-estimated
1	Previous Interval Volume				Previous Interval Occupancy				Previous Interval Speed			
2	0											
3		0 – 10										
4		0 – 500										
5		0 – 800										
6			50 %									
7				50 %								
8					0							
9						0 – 1						
10						15 – 30						
11							30 %					
12							40 %					
13								40 %				
14									0			
15										0 – 5		
16										0 – 30		
17											25 %	
18											30 %	
19												30 %
20												60 %

Table 4.1 The experimental scenarios

5. OFF-LINE SIMULATION EXPERIMENTS

The off-line test involves the evaluation and benchmarking of various components of the Internet-based traffic control architecture in an off-line mode. Because of the non-availability of traffic-related data from the real network, a traffic simulation algorithm, namely DYNASMART, is used as a proxy for the real world traffic system to generate traffic data as input to the on-line control architecture. Both sequential and parallel experiments were conducted on SCORCH-IT as part of the off-line tests.

5.1 Off-line Test Experiment Description

To test the validation and performance of the Internet-based on-line traffic control architecture, a simulation environment, shown in Figure 5.1, was constructed.

In the off-line test architecture, two main components are involved. A traffic simulation model is embedded on a PC or workstation as a proxy for the Borman network. The network structure and the real-time O-D demand data for the Borman network are fed to the simulation model. It generates data such as time-dependent link travel time, vehicle position information, and link traffic volume. This data is transmitted to SCORCH-IT located in the Traffic Operations Laboratory at Purdue University through an intranet connection with 10Mbps bandwidth. A virtual simulator and a route guidance algorithm are embedded on SCORCH-IT, which uses the virtual real-time traffic information data as input, and generates traffic control strategies for network control. These strategies are implemented using the same connection on the proxy network.

The test network is located in northern Indiana and includes the Borman Expressway and the surrounding arterials and street network. The Borman Expressway network is represented as a grid structure containing 197 nodes and 458 links, and is divided into 14 zones. With real-time information, drivers are assumed to obtain en-route updates on the best route to their destinations at every decision point. The traffic network link characteristics data (such as number of lanes, length, capacity, and free flow speed) were collected from relevant state and local transportation agencies. The maximum bumper-to-bumper and jam densities for all links are assumed to be 260 vehicles/mile and 160 vehicles/mile, respectively. Since a significant amount of the Borman Expressway traffic are large trucks, local arterials serve as detours for passenger cars and vans. Hence, it is reasonable to include all roads in the vicinity of the Borman Expressway in the simulation network to test the system performance.

5.2 Data Communication Tests

Though the virtual network and SCORCH-IT are connected by a 10Mbps intranet, the actual transmission speed is subject to some randomness caused by communication and local-area network (LAN) related issues. In order to get a robust estimate of the performance of the architecture, the associated experiment was conducted at different times of the day and on different days. Figure 5.2 shows the results of a 10MB-file transmitting experiment between the virtual network and SCORCH-IT. The experiment was conducted continuously every half-hour at different times of the day during a 3-day period. Data shows that the actual communication speed varies from about 0.5Mbps to 4Mbps.

In the off-line test of the online control architecture, the size of data (input data) transmitted from and to SCORCH-IT is usually about 2MB. Thus the communication time of the online control architecture varies from 5 seconds to 30 seconds in each direction depending upon the prevailing communication conditions.

5.3 Sequential Algorithm Tests

Inside the computational unit (SCORCH-IT), the performance of the integrated control strategy algorithm was tested. The tested components include the traffic simulation model, DYNASMART, which is combined with a Dynamic Traffic Assignment (DTA) algorithm called the Multiple User Classes Time-Dependent Traffic Assignment (MUCTDTA). They are embedded on SCORCH-IT, which works as a computational unit to determine the real-time traffic control strategies for the virtual network.

MUCTDTA, illustrated using a flowchart in Figure 5.3, provides optimization solution strategies to the on-line architecture. In a future ATIS/ATMS scenario, in order to optimize the performance of a network through the provision of real-time routing information to different motorists, some factors such as information availability, information supply strategy and driver response behavior must be taken into consideration for different user classes [8]. In particular, two user classes are incorporated into the formulation of the tested algorithm: (i) equipped drivers, who follow prescribed system optimal paths (called user class 1 or SO class) and (ii) equipped drivers who follow user optimum routes (called user class 2 or UE class). Other driver classes may be considered without substantial influence on the computational burden of the algorithm.

The MUCTDTA algorithm uses information about every trip maker in terms of his/her origin, destination, start time and user class for the entire assignment horizon, to develop an integrated scheme that assigns to each user a path to his/her destination so as to achieve some system-wide objective. DYNASMART is a fixed time-step mesoscopic simulation model and provides link, path and vehicle information to the MUCTDTA algorithm.

The simulation period used in the tests is set to 60 minutes, and the traffic demand is generated over 30 minutes. The total number of vehicles generated during the period of interest is 14153 vehicles. The time-dependent origin-destination demand data on the Borman Expressway is used to simulate the traffic network.

The CPU time of each part of the algorithm is observed. As illustrated in Figure 5.4, the simulation component (PARTCO) of the model takes the maximum amount of computational time.

5.4 Parallel Algorithm Tests

According to the benchmark for the sequential tests, the MUCTDTA algorithm (excluding the traffic simulation model) can be split into three major parts. After inputting the basic data such as network information and travel demand information between all origin-destination pairs, the path strategies for SO and UE users are computed separately. Then, all vehicles in both user classes are sorted together according to the time and the link on which they are generated. There is no communication between the SO and UE parts; so their execution is independent of each other. The sorting algorithm is executed after the SO and UE components. There are two main subtasks in

the SO algorithm: a module called SOMARGINAL, which computes the marginal travel time on each link; and a module called SOPATH, which computes the time-dependent shortest paths for the SO class vehicles. These two components are executed sequentially and account for over 80% of the total CPU time of SO.

In terms of the parallelization of the traffic simulation model, we focus on two key subroutines. The first module called NODETRAN is used to update information on vehicles that need to cross an intersection during the current simulation interval. It considers all the vehicles reaching their downstream nodes and updates the relevant network and vehicle related parameters. The second main subroutine called LINKTRAN updates the prevailing vehicle information on each link. It determines the number of vehicles that enter the network at the current simulation interval, updates the vehicle positions and checks if vehicles arrive at destinations.

5.4.1 Parallelization Paradigms

Different programs and subroutines are integrated together to execute the MUCTDTA algorithm and the simulation model. The integrated algorithm contains both parallelized and sequential subroutines. Two basic parallel paradigms were employed to integrate the algorithm.

In Flynn's hardware taxonomy, computer hardware is classified as SISD (Single-Instruction-Single-Data, which is the traditional sequential computer), SIMD (Single-Instruction-Multiple-Data) and MIMD (Multiple-Instruction-Multiple-Data). Because of the flexibility and variety of modern parallel computer structures and the powerful functions provided by parallel software tools, this classification is no longer restricted to

the hardware classification, but can also be applied to the parallel programming paradigms and parallelization methodologies.

To ensure the correctness of results, parallel programs must effectively coordinate more than one process. Different processes may contain identical or different instructions and commands. When more than one flow of control is supported at the same time by the parallel program, the program is referred as a control-flow program. In a message-passing paradigm, it is also referred as the MIMD model, which means that many processors can simultaneously execute different instructions on different data. MIMD is the most general form of parallel programs. Each processor runs under the control of its own instruction sequence, but they are not totally independent because they may access or modify the same copy of data. In MIMD, synchronization is enforced by certain synchronization mechanisms, usually a lock, which permits only one process to access any piece of data at an instant. MIMD is best suited to medium or large-grained parallel problems.

In a data flow program, the data is partitioned into separate subsets and then identical operations are performed on the different sets concurrently. Therefore, each process executes exactly the same instruction at the same time but on different data. This model is also called SIMD. In a SIMD paradigm, data is usually distributed using a message passing paradigm. The execution of all processes is tightly controlled by lock-step unison so that synchronization is guaranteed.

In many cases, processes do not necessarily execute exactly the same instructions at every time step after the particular pieces of data are distributed. Thus, a model called SPMD (Single-Program-Multiple-Data) is also useful. The major difference between SPMD and SIMD is the degree of synchronization. SIMD is an example of synchronous

data parallel computing and SPMD is its related asynchronous version. In SPMD the same program is run but with different data and unlike the MIMD paradigm, the instructions are not synchronized at every time step.

Two parallel programming paradigms, namely MIMD and SPMD were implemented in the parallelization of the MUCTDTA algorithm. Because the computation of the SO marginal times and the two path processing algorithms decide the overall performance of the problem, the parallelization methodologies employed will be classified by the way SO and UE components are arranged.

When MUCTDTA is implemented according to the SPMD paradigm, the algorithm is run in the same order as the sequential algorithm. SO, UE and the sorting algorithms are executed one after another. The parallelized parts are computed using the 16 processors available on SCORCH-IT. At any instant, the processors run the same program, even though they may not be synchronized for every instruction in the program. The data is exchanged by a message passing paradigm realized by calling MPI operations in the programs. The flow chart of the SPMD paradigm for MUCTDTA algorithm is shown in Figure 5.5 (a).

When the MIMD paradigm is implemented, the SO and UE algorithms are executed at the same time but on different processors. This is a large-grained parallelization and is efficient because the two parts are completely independent. SO and UE modules are further parallelized and the available processors are assigned according to the expected time of execution. In the flow chart of the MIMD paradigm for the MUCTDTA algorithm shown in Figure 5.5 (b), m processors are used to execute the SO algorithm and n-m processors are used for the UE algorithm, where n is the total number

of processors available. After the completion of data input, the processors are divided into two groups to be assigned for the UE and SO algorithms.

5.4.2 Parallel Test Results

Both SPMD and MIMD paradigms are implemented for the parallel MUCTDTA algorithm using 16 processors on SCORCH-IT. From Figures 5.6(a) and 5.6(b), it can be seen that although MIMD has lower execution times, SPMD is more efficient due to its higher speedup values for both parameter sets. This is primarily due to the increased number of processors used in the MIMD paradigm. In the MIMD paradigm, when all 16 processors are used for a single algorithm, some processors will be idle due to the heterogeneity in assigned tasks and the processor capability. In the SPMD paradigm, processors will be much busier because lesser number of processors used for each algorithm ensures that efficiency is enhanced. Figure 5.7 shows the CPU time of the main parallel programs (marginal time algorithm, SO and UE path processing algorithms and vehicle sorting) executed on different numbers of processors. It can be seen that beyond 10 processors, the benefits due to additional processors becomes marginal. Therefore, depending on the trade-off between the execution time and efficiency, an optimal number and an appropriate paradigm should be used.

5.5 Future Real-time Tests

Upon availability of the field data from the Borman network, real-time tests for the online traffic control architecture will be conducted. One of the main differences between off-line tests and real-time tests is that the test network in the off-line tests will be replaced by the real traffic network. Traffic data from the field will be transmitted

from the TCC to SCORCH-IT. Another important difference is the data communication. In addition, the connection between the TCC and SCORCH-IT will take place via the much slower internet as compared to the high-speed network that was used in the off-line tests. There are various internet connection technologies that are currently available in the internet service provider (ISP) market. These connections have different characteristics with respect to speed, cost, and reliability. A judicious choice of the connections is critical to the efficient functioning of the on-line architecture. Table 5.1 shows some possible choices and their important attributes.

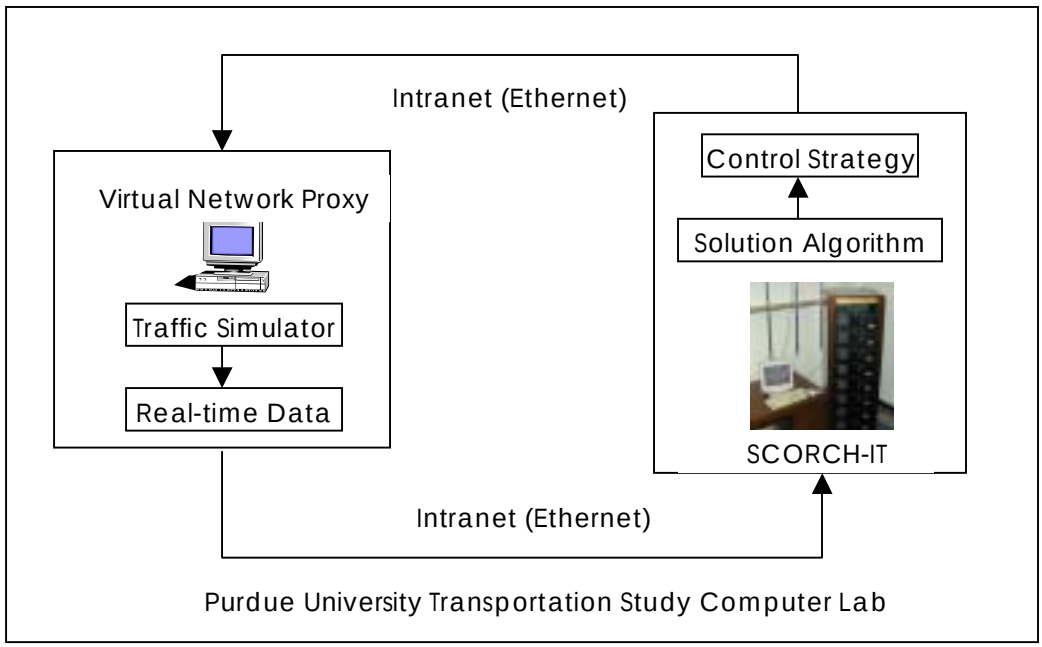


Figure 5.1 Off-line Test Architecture

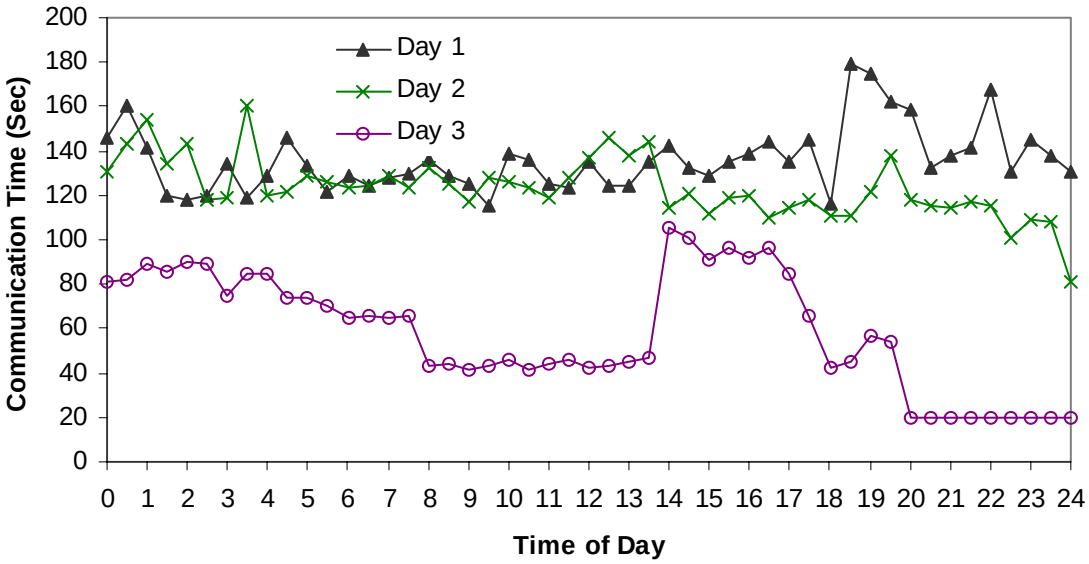


Figure 5.2 Data Communication Tests

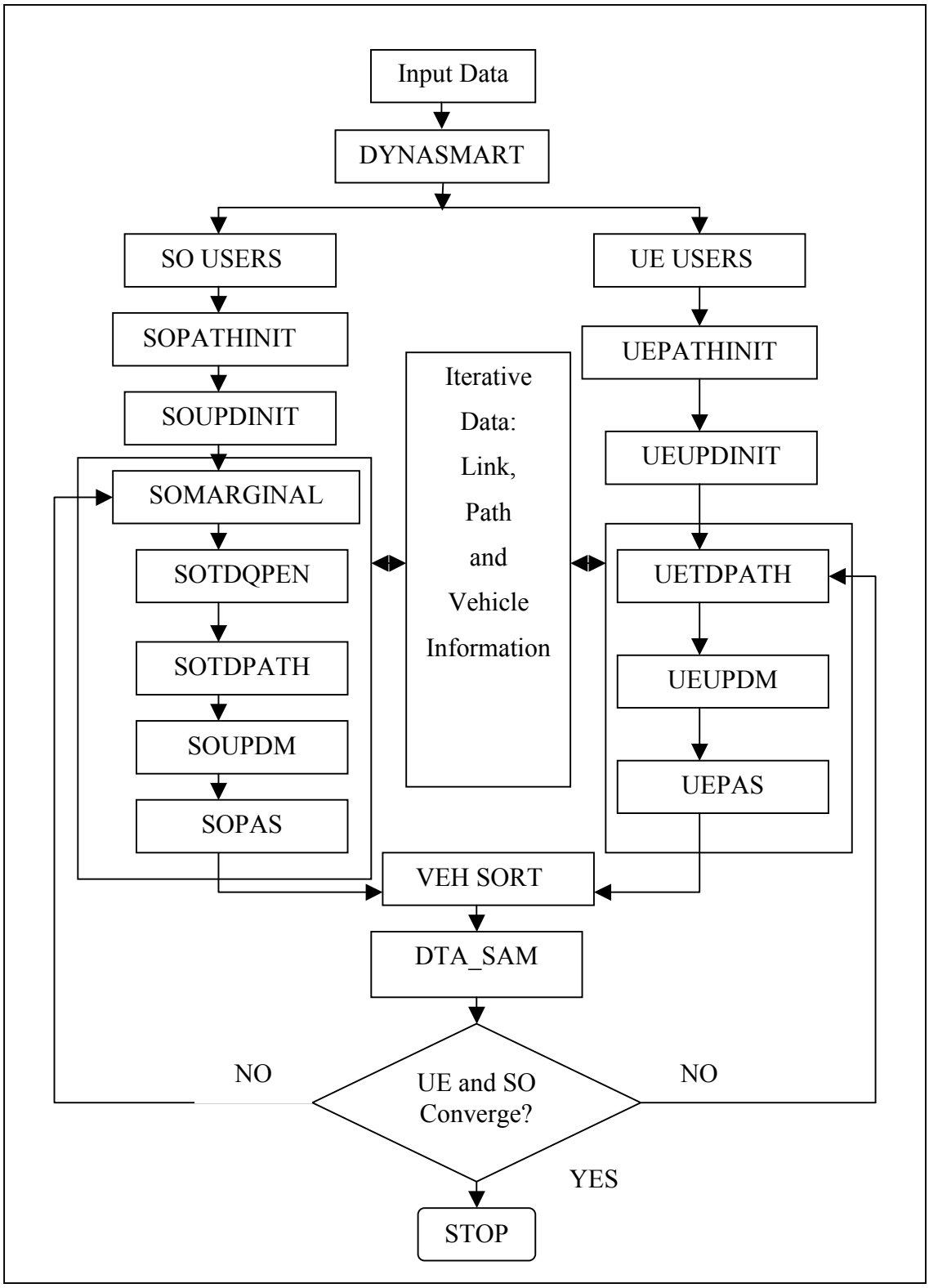


Figure 5.3 Flow Chart for MUCTDTA Algorithm

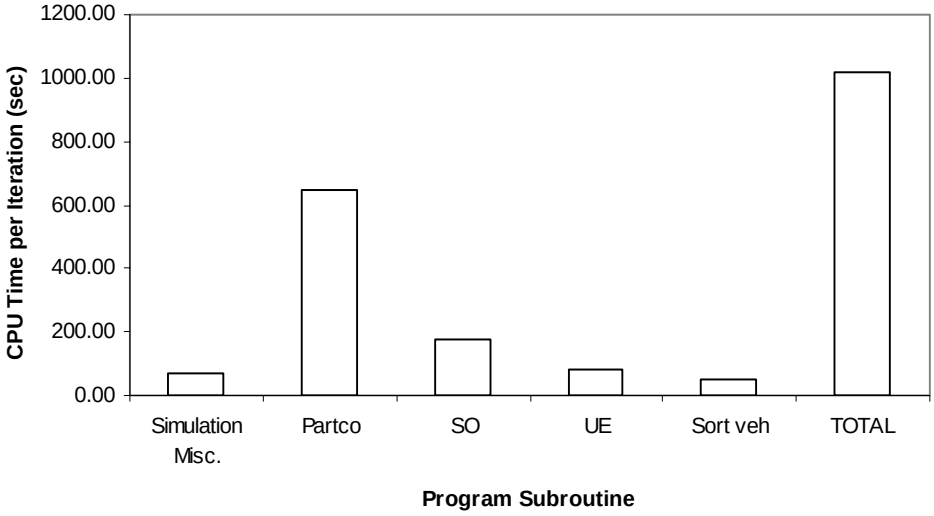


Figure 5.4 CPU Time for Sequential MUCTDTA Algorithm

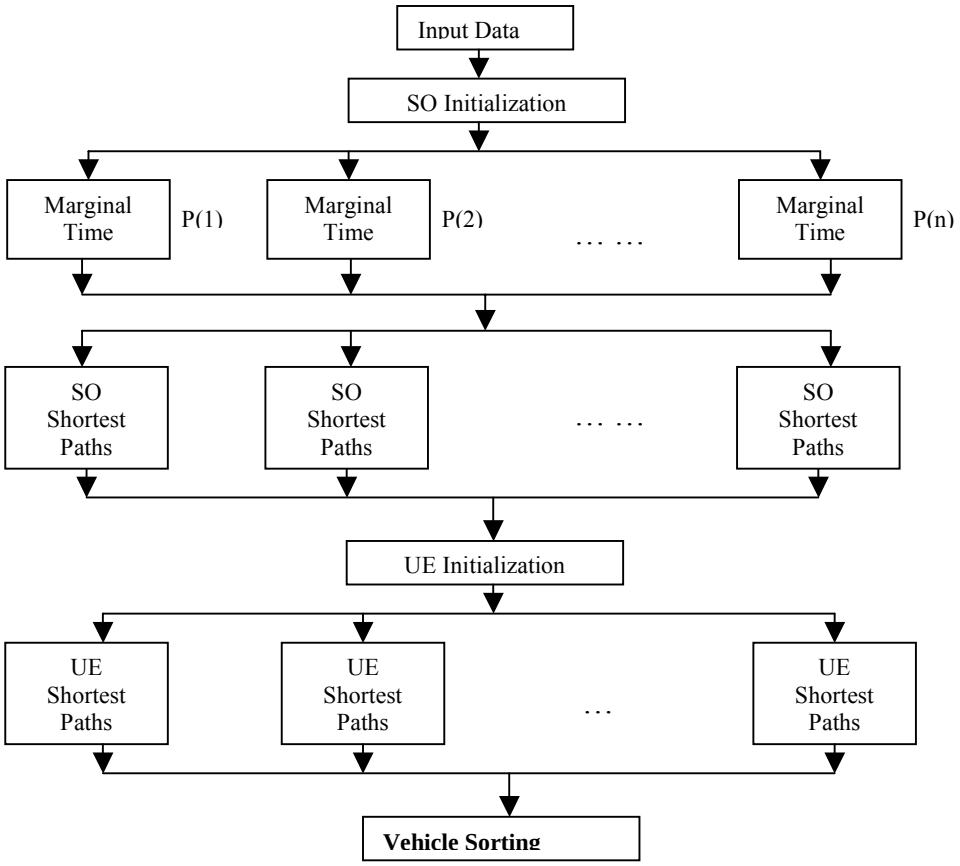


Figure 5.5(a) SPMD Parallel Programming

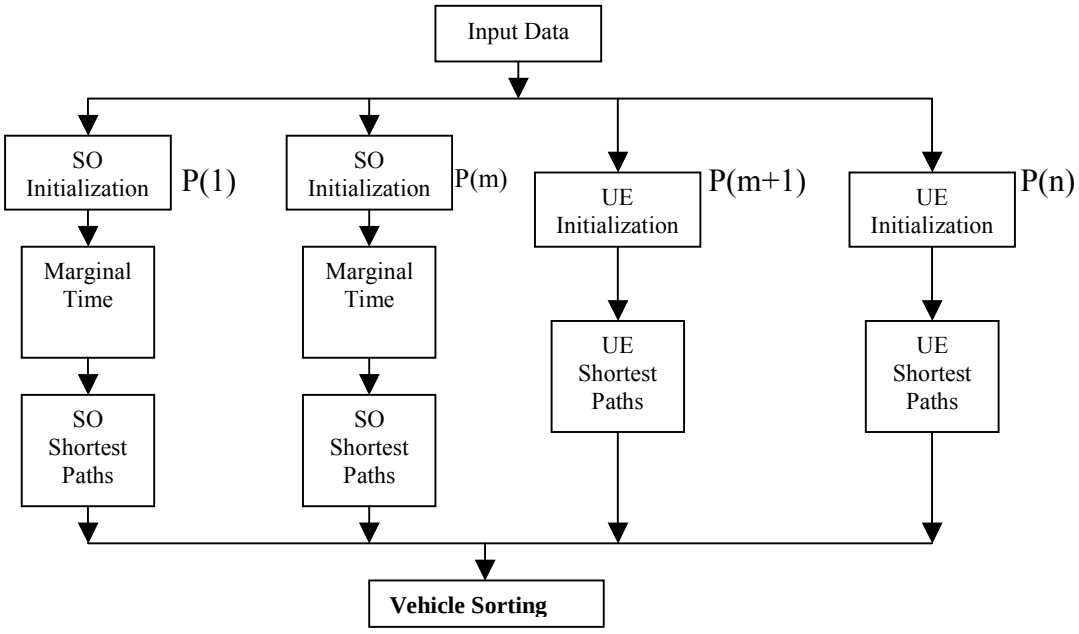
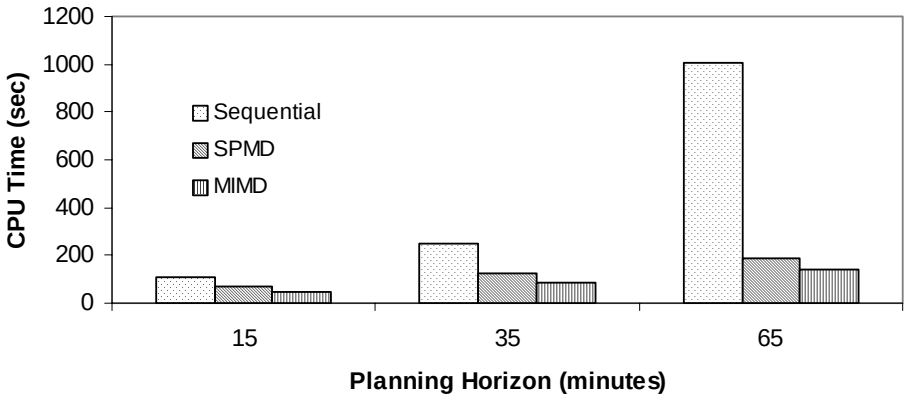
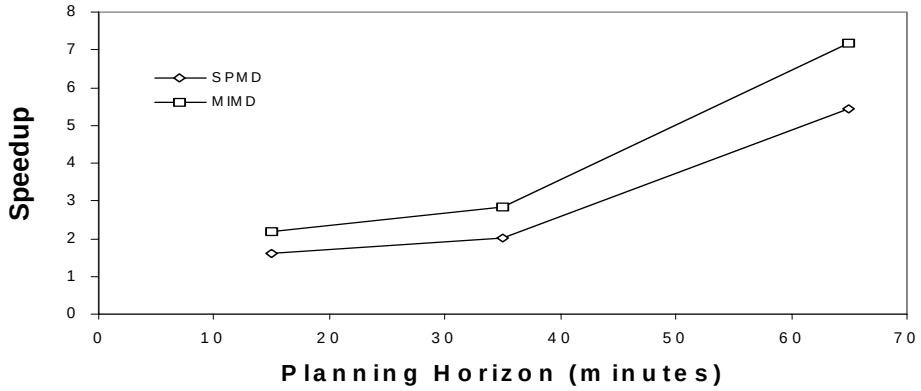


Figure 5.5(b) MIMD Parallel Programming

Figure 5.5 Flow Chart of MUCTDTA Algorithm for SPMD and MIMD Paradigms



(a) The Computation Burden for Different Planning Horizons



(b) Speedup for Different Planning Horizons Using 16 Processors

Figure 5.6 Parallel Performance for MUCTDTA Algorithm

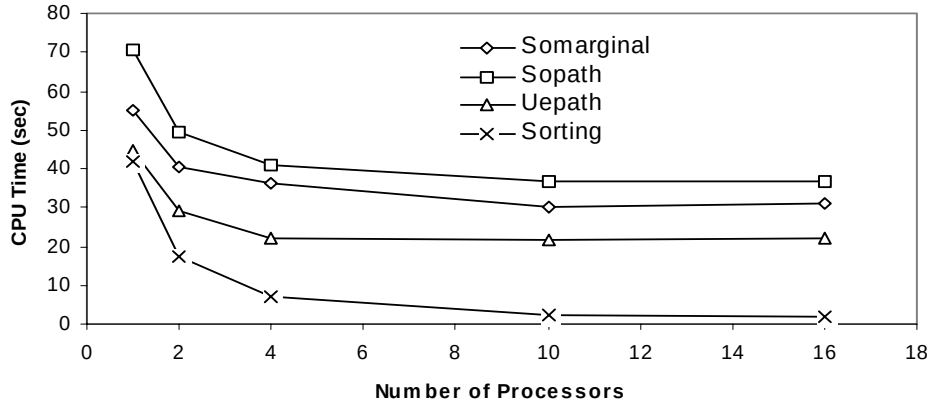


Figure 5.7 CPU Time for Main Parts of Parallel MUCTDTA Algorithm

Internet Connections Services	Satellite	ISDN	ADSL	Cable Modem
Approximate Installation and Hardware	$1200	$400	$100 - $500	$100
Approximate Recurring Cost (monthly payment)	$330	$300	$60-$100	$50
Speed	256Kbps downstream, 128Kbps upstream	128Kbps downstream and upstream	640Kbps or 1.6Mbps downstream, 90Kbps upstream	10Mbps downstream, 768Kbps upstream
Mobility	Good	No	No	No

Table 5.1 Comparison of Some Internet Connection Services

6. CONCLUSIONS

An internet-based on-line architecture for real-time traffic systems was presented. The proposed architecture incorporates the internet and the Beowulf computing paradigm to efficiently address issues arising in the evaluation and control of real-time traffic systems equipped with ITS technologies. It explicitly accounts for the calibration and consistency checking needs of the models being used within the architecture. The architecture incorporates fault tolerance methods for errors encountered at the architecture level and due to the malfunctioning of the sensors collecting field data. The architecture is also designed to be remote in that it can serve multiple spatially separated traffic control centers by allowing the models and the databases to be distributed. Furthermore, the framework is not model-specific implying that the route guidance, calibration and fault tolerance models can be upgraded as newer models become available.

The Beowulf Cluster is proposed as the computing paradigm within the on-line architecture. Beowulf Clusters serve as a viable alternative to expensive customized architectures in satisfying the heavy computational needs of real-time traffic systems. Recently, there has been an enormous amount of interest amongst researchers and practitioners from various fields in the concept of the Beowulf Cluster as a cheap means of achieving high performance computing. The idea has been applied successfully in a wide variety of areas including Earth and Space Sciences, Molecular Dynamics Simulation in Metals and Computational Fluid Dynamics. While typical supercomputers

cost between $ 500,000 and $ 1,000,000, a Beowulf Cluster of comparable performance can be built for less than $ 15,000. Besides providing super-computing power at affordable costs, the Beowulf Cluster provides the flexibility to design asymmetric architectures optimized to the problem being solved. This report discussed the architecture and components that comprise a typical Beowulf Cluster along with the detailed specifications of an implementation called SCORCH-IT.

A judicious investment of resources is critical to the success of the Beowulf Cluster. The hardware used in the Cluster is optimized for the problem being addressed, and a thorough understanding of the application will ensure efficient investment of available resources. For instance, the experiments on the mini version of SCORCH-IT had indicated that gains in execution times of SPA can be expected with an increase in the number of processors. In addition, the need for the simultaneous execution of other tasks such as data retrieval and fault tolerance as part of the on-line architecture led to the expansion of SCORCH-IT into a 16-processor Beowulf Cluster. Recent experiments have shown that increased memory in some nodes may facilitate a more optimal I/O scheduling potentially leading to further gains in overall execution times. Multi-threading is the next step in order to facilitate communication between processors in a machine through shared memory. This may result in additional speed-ups especially for longer periods of interest and/or larger networks where the data files can be substantially larger. Future work will focus on carrying out similar experiments for other modules in MUCTDTA in order to gain insights into how resources can be optimally invested in the expansion of SCORCH-IT.

Off-line simulation tests revealed the effectiveness of the on-line architecture and the computing paradigm that has been proposed. The tests showed that the framework can effectively address medium-sized networks such as the Borman expressway. Depending on the specific problem being addressed the architecture of the Beowulf Cluster and the functions performed by the nodes within the cluster may have to be altered. However, the off-line tests used fast Ethernet connections for data transmission thereby avoiding a number of important communication related impediments that would arise in a real-world scenario. Carefully designed on-line experiments using actual sensor data and field connections may be critical to identify all the communication and computational issues that are likely to arise if the framework is to be deployed.

LIST OF REFERENCES

[1] Sterling, T. L., Salmon, J., Becker, D. J. and Savarese, D. F. (1999), *How to Build A Beowulf: A Guide to the Implementation and Application of PC Clusters*, MIT Press, Cambridge, MA.

[2] Snir, M., Otto, S., Huss, S., Walker, D. and Dongarra, J. (1999), *MPI – The Complete Reference: Volume 1, The MPI core*, 2nd Edition, MIT Press, Cambridge, MA

[3] Kirch, O. (1995), *LINUX – Network Administrator's Guide*, O'Reilly

[4] Ziliaskopoulos, A. and Mahmassani, H. S. (1994). "A Time-dependent Shortest Path Algorithm for Real-time Intelligent Vehicle/Highway Systems Applications", *Transportation Research Record* 1408, Washington, D.C., pp. 58-64

[5] Peeta, S. and Chen, S. C. (1999). "A Distributed Computing Environment for Dynamic Traffic Operations", *Computer-Aided Civil and Infrastructure Engineering*, No. 14, pp. 239-253

[6] Kuck, D. J. (1996). *High Performance Computing*, Oxford University Press, New York

[7] Ramsden, E., Data Acquisition System Architecture, *Machine, Plant & System Monitor,* pp. 10-13, May/June 1999

[8] Lewis, T.G. Foundations of Parallel Programming, a Machine-Independent Approach. IEEE Computer Society Press, Los Alamitos, 1994.

[9] Mahmassani, H.S., Y. Hawas, K. Abdelghany, Y. Chiu, A. Abdelfatah, N. Huyhn, and Y. Kang, DYNASMART-X: System Implementation and Software Design, Technical Report ST067-085-Volume III, 1998.

[10] Krogmeier, J.V., Sinha, K.C., Fitz, M.P., Peeta, S., Nof, S.Y. (1996), "Borman Expressway ATMS Equipment Evaluation", JTRP Final Report FHWA/IN/JHRP-96/15, West Lafayette, IN.

[11] Cartwright, M. (1990), "Fourier Methods for Mathematics, Scientists and Engineers", E.Horwood, New York., NY.