INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Université d'Ottawa · University of Ottawa

Study of Central Control Actions

in

Computer Simulation

By

Yang Zhang

A Thesis Presented to the University of Ottawa in Fulfillment of the Thesis Required for the Degree of Master of Science (Systems Science)

OTTAWA, OCT. 2001

Acknowledgements

I extremely appreciate Professor Francois M. Julien, who has supervised me in my thesis work, for his generous support, academic direction, enthusiastic encouragement and patience with me.

I extend my gratitude to Mr. Tom Moynihan and Mr. Gord English for their support, suggestion, guidance, and patience.

Propound thanks to Dr. Saeed Zolfaghari for his generous cooperation and constructive ideas.

Thanks to Mr. G.D. Dean, Superintendent Transportation Operations in OC Transpo, for his generous donation of time to familiarize us with the operations of the control centre.

Thanks to the members of the thesis examining committee, professor Dan Lane and Jeff Sidney, for the constructive feedback.

Dedication

To my dear Mom and Dad, husband, and all family of my brother.

Table of Contents

Table of Figures

ABSTRACT

The Canadian government identified the reliability of public urban transit systems and the improvement of customer services as a priority. Therefore, in 1997, Transport Canada and the Regional Municipality of Ottawa-Carleton have set up a project called the "Bus Priority System Project" to enhance the capabilities of the existing OPSSIM simulation model in order to satisfy these requirements. The enhancement of the simulation model would allow for the training of controllers, as well as for the evaluation of new control strategies.

This thesis describes a part of the project. It works on the design, research, investigation and implementation of a Central Control Action Module, and on the design and implementation of a user-friendly Run Time Bus Progress Screen.

A brief user guide describing how to use these new features is also included in the thesis for the benefit of potential users.

In the conclusion, overviews for future research are identified.

I. INTRODUCTION:

The improvement of customer service and reliability for public urban transit systems has been identified as a priority by many countries in the world. At the national level, Transport Canada, a department of the Government of Canada, promotes studies to enhance technology, strategy and tactics in the urban transit area. In Ontario, the provincial government, local governments and various transportation properties such as OC Transpo in Ottawa, participate in various urban transit improvement projects as well. In the US, it is worth noting the work done by, and promoted by, the Department of Transportation of the United States Federal Government. Many of its initiatives (often in conjunction with university or private researchers) are reported in public documents, and provide an excellent source of information about urban transit research efforts around the world.

Locally, OC Transpo, the urban transit property, which serves the Ottawa-Carleton Region, has participated in and supported a wide range of initiatives. Included here is a multi-year simulation study of bus movements. The study, which is focused on service improvement through bus movement control strategies, otherwise referred to as bus scheduling strategies, is being led by the consulting group TranSys Research Ltd. of Kingston, and supported by OC Transpo, and Transportation Canada. Currently, the simulation model, which was created several years ago, is being expanded to facilitate study of a wider range of simulation studies.

In this thesis, we present expanded research on the existing computer simulation model OPSSIM including the design and implementation of one central control action, the Insert Bus Following Breakdown action, and of a run time bus progress screen, as well as the verification and validation of the expanded system.

The progress of buses along their routes may be quite variable due to traffic and weather conditions, variations in the number of passengers boarding and alighting, and other factors. As a result, riders may experience service which varies from excellent to poor (e.g., early or late busses, longer trip times than scheduled, crowded buses, missed transfers, etc.). With the advent of modern information and communication systems, urban bus systems now have capabilities for the real-time control actions for service improvement. Examples of such actions are real-time addition of extra buses to deal with an unexpected accident or breakdown on a route, either speeding up, or, delaying buses to even out the spacing between buses, or, intentional skipping of stops in certain situations, etc. However, the system quickly becomes very complex to manage and to model. Our modeling efforts have been focused on real-time control for service improvement on a single route.

Two critical existing systems, which are the Automatic Vehicle Location and Control (AVLC) system and the Automatic Passenger Counting (APC) system, provide input data to OPSSIM. These data provide necessary and valuable information regarding arrival and departure times of buses at each stop, the number of passengers boarding at each stop, and schedule

adherence. A lot of work in implementing the real-time control actions is done with the AVLC system, so, before discussing the details of our implementation, we briefly describe OC Transpo, the AVLC and APC systems, and the OPSSIM simulation model.

I-1. OC Transpo with two data systems:

OC Transpo is a public urban transit property that provides transit service to the communities of Ottawa, Nepean, Vanier, Rockcliffe, Gloucester, Kanata, and Cumberland. Some OC Transpo routes also operate to downtown Hull, Quebec. The service area covers more than one hundred routes and about five thousands stops at present [4]. OC Transpo has been active in innovating various measures to facilitate service improvement. Examples, already mentioned, are the Automatic Passenger Counting (APC) system, the Automatic Vehicle Location and Control (AVLC) system, and a simulation system (OPSSIM). All information and data gathered by the first two systems have been very valuable for evaluating the performance and service reliability of the system, and also for the modeling efforts that went into the design of the OPSSIM simulation model.

Because most of our research is based on the information and data given by the APC and AVLC systems, we describe the two systems in briefly below.

APC System: The Automatic Passenger Counting (APC) system was first installed at OC Transpo in 1976, progressively updated in the 1980s

and is now well developed. The system can generate basic information on vehicle location, the number of passengers boarding and alighting, passenger loads and travel time between stops [3]. All of this information is gathered in the form of off-line, report-oriented information that can be used for transit planning and management, service analysis, and marketing. The data given by the APC system are not real-time oriented, but the system is still useful as one planning tool for improving bus service.

AVLC System: The Automatic Vehicle Location and Control (AVLC) system was implemented in the early 1990s [3]. The system can provide real-time data and information on bus performance and service reliability and also offer new opportunities for bus service analysis. The main functions of the system are to monitor and trace the location of all buses at strategic points throughout the service region, and to display incidents, or deviations between the actual and the planned flow of buses to controllers. Controllers and drivers are also to engage in two-way communication.

The data files given by the two systems during a selected period compose the input data file system supporting the OPSSIM simulation model (OC Transpo Transit Simulator). These input data files that are used in this thesis will be described in Chapter III.

I-2. Simulation model:

" OC Transpo Transit Simulator OPSSIM version 1.00 " is a simulation model developed by the Canadian Institute of Guided Ground Transportion (CIGGT) for OC Transpo in September 1992[1]. During the

period from 1992 to 1993, it was modified four times. The modification history is listed in the following table [1]:

Date	Version	Description
Sept 23, 1992	X1.00	Initial Release to OC of File Version
Dec 9, 1992	X1.01	ReRelease to OC to meet DDE Requirements of File Version.
Mar 9, 1993	X1.10	New Release with code for Integrated Version.
Sept 10, 1993	X1.20	ReRelease to bring CODE and DDE up to OC Standards.
Dec 1, 1993	X1.30	Removal of Hardcodes Filename and rewrite of the '4' DDC Docs.

Table 1: The history of OPSSIM versions

OPSSIM (version 1.00) is the validated simulation model for the OC Transpo Route 95. The model can simulate, *for only one route at a time,* the arrival and boarding distribution of passengers, bus departure from, and arrival at a stop, and the behavior of buses and riders associated with various schedule control strategies (actions which can be taken to correct problems arising along the route). It was later enhanced to deal with headway control strategies [1]. OPSSIM could be used to simulate the operation of other routes (one at a time) if the appropriate data were prepared, as has been done for Route 95.

Although OPSSIM (version 1.30) proved to be useful for researchers to do simulation experiments, and for OC Transpo as a training tool for new central controllers, there were also some limitations. The most notable is that it can not deal with on-line control policies. OPSSIM (version 1.30) did not have an interface between the run time simulation and the control module, which allow the user to insert control policies to impact the running bus performance, a consequence of which is the inability to implement real-time intervention as a human controller would do. Our expanded research will provide the extra software to make OPSSIM overcome this weakness.

Currently, Transys Research Ltd. (a transportation consulting company based in Kingston, Ontario) is updating OPSSIM to enable it to perform multi-route simulation using traffic signal control strategies, and also real-time bus control strategies based on on-line control actions. It is the last type of control strategy that is the focus of much of this thesis. A new version of OPSSIM will be generated in part from the developments made in this thesis.

I-3. Literature Review:

The literature review is divided into three sections. The general review given in section I-3-1 relies on two extensive and comprehensive review documents produced by the U.S. Department of Transportation. The relevant sections for our research are described below. In section I-3-2, we review previous work done locally at the University of Ottawa by two previous master students. The current research is an extension of this work.

In the last section, we introduce the terminology and basic concepts relative to public urban transfer strategies.

I-3-1. Review documents from the U.S. Department of Transportation:

Two excellent review papers posted in the web by the U.S Department of Transportation are very useful sources of information on new strategies and new technologies, implemented or proposed, for urban transit improvement throughout the world.

The paper entitled "Advanced Public Transportation System: The State of the Art Update '96 "[7] documents an investigation of the extent of adoption of advanced technology in the provision of public transportation service in the U.S. and Canada. Although it is not an exhaustive review, it covers the key initiatives in most major urban areas. It focuses on some of the most innovative or comprehensive implementations of new technology approaches, categorized under four types of services/technologies: fleet management, traveler information, electronic fare payment and transportation demand management. Among the above four types of services/technologies, fleet management compares most closely to the research herein.

Fleet management incorporates many of the vehicle-based Advanced Public Transportation System technologies and innovations for more effective vehicle and fleet planning, scheduling, and operations. It focuses on

the vehicle, improving the efficiency and effectiveness of the service provided, and on passenger safety. Some major technologies and innovations used in fleet management in North America are communications systems, geographic information systems, automatic vehicle location, automatic passenger counters and transit operations software. Currently, these technologies have been applied in many cities and regions in North America, and have played an important role for improving the reliability of public transportation service.

The installation of advanced communication systems in public transportation is a prerequisite for the implementation of a real-time bus controlling strategy. The link between the bus and control centre is the most critical part in the communication system. Drivers and centre controllers communicate with each other using the bus/control centre link. The central controller can obtain on-line information, such as bus progress, traffic, bus location, and passenger load. Controllers can also send online control directives to bus drivers.

Geographic information systems (GIS) are designed to support the capture, management, manipulation, analysis, modularity and display of spatially referenced data[7]. They are useful for dealing with complex, real-time and planning issues in operation planning and other management and operational needs. These include bus route design, street planning, design of parking and other facilities, rider loading, running times, scheduling, bus assignments, accidents, and customer complaints - for service and facilities

planning; street and route maps, service performance monitoring, vandalism location and history, and emergency call location identification.

Automatic vehicle location systems (AVLS) are computer-based vehicle tracking systems[7]. These systems operate by measuring the actual real-time position of each vehicle, and relaying the information to a central location. Benefits of the use of AVLS in public transportation systems include[7]:

- increased overall dispatching and operating efficiency
- more reliable service, promoting increased ridership
- quicker response to service disruptions
- inputs to passenger information systems
- increased driver and passenger safety and security
- quicker notice of mechanical problems with the vehicles, reducing maintenance costs
- inputs to traffic signal preferential treatment actuators
- more extensive planning information collected at a lower cost than manual methods

As a result of these benefits, AVLS's are used more and more. According to the investigation by U.S. Department of Transportation, between 1992 and 1996, AVLS use in U.S. transit systems has increased more than 100 percent[7]. There are at least 58 AVLS's in operation, under installation, or planned in the U.S. and at least six in Canada[7]. At a minimum, each includes a specific location technology and a method of transmitting the location data from the bus to dispatch. Usually, each system still has its

own character. For example, in Atlanta, Georgia, the Metropolitan Atlanta Rapid Transit Authority (MARTA) is installing a TMS GPS-based system on 250 of its approximately 750 buses. The AVLS will be installed into both their existing CAD (Computer Aided Dispatch) system and bus information kiosks located at MARTA rail transit stations. In addition to kiosks, AVL information will be sent to 15 new passenger information devices at busy bus stops, and 100 vehicles will be equipped with audible "next-stop" enunciators and visual, digital "next-stop" signs. There will be Automatic Passenger Counters (APC) on 15 of the buses, although the passenger load information will be used for planning purposes only, and will not be available in real time. MARTA is also hoping to link the AVLS into a separate signal preemption project. Finally, the system will include a fiber-optic link to the Georgia DOT's (Department of Transportation) advanced transportation management system. This will be a multi-modal, multi-jurisdictional system, including complete transit and traffic information, video feeds, and trip itinerary planning. The list of known existing or planned transit AVL systems in North America is given in reference [7]. The AVL system of OC Transpo is included in the list.

Automatic passenger counters (APC) are a well-established, automated means for collecting data on passenger boarding and alighting by time and location[7]. These data may be used for a number of applications, both real-time and delayed. Currently planned APC installations are most commonly implemented in conjunction with AVL systems, in order to take

advantage of the available location information. Known North American applications of automatic passenger counters are listed in the reference [7].

All four of these technologies have been used in our research. AVL and APC systems in OC Transpo provide the real data resource for our research. A communication system and a GIS provide good references for our design of a user-friendly interface, which will provide real-time information on the current state of the transit system.

Another paper entitled "Adaptive Control of Transit Operations" [6] introduces some already well-developed methods and models for controlling the movement of transit vehicles along their routes through adaptive control. The basic concept in the paper is that adjustments in traffic signals and other controlled variables, based on real-time information, may be used to help transit vehicles move at higher average speeds and to better adhere to schedules. The quality of bus transit service concerns both operators and users. Bus routes may have many stops and traffic signals that affect bus movements and operational efficiency. The paper poses that the major reason for controlling bus movements along high frequency bus routes is the preservation of even headways. One natural tendency of buses traveling along a route is bunching up into "platoons". This occurs because variations always exist in bus dwell times at stops and in travel times along the routes. If a bus falls slightly behind schedule for any reason, it will have more than the average number of passengers to pick up at subsequent stops, which results in further delays and even larger abnormal loads downstream. As a result, such a bus falls further and further behind

schedule. Conversely, the following bus would have fewer passengers than usual and lower dwell times, allowing it to catch up with the preceding one. Therefore, authors propose that potential control strategies should be focused on reducing one or more sources of headway variability. In a broad sense, the major objectives of control strategies are to keep platoons from forming or to break them up after they have formed, and to ensure proper arrival times at transfer points.

An adaptive control system should predict negative conditions well ahead of time and implement the desired adjustments early rather than wait until the last moment, when options may be very limited. Nonetheless, an effective system should also be able to respond quickly to any new information, including major surprises. Thus, an ideal adaptive control system should have very good predictive capabilities as well as data collection, data processing, decision making and communication capabilities[6].

After analyzing four classes of strategies (reduction of the number of stops, signal preemption, provision of exclusive right of way, and vehicle dispatching controls) that can be used for improving the reliability of bus transit service, Guey-Shii Lin, Ping Liang, Paul Schonfeld and Robert Larson[6, 1995] developed two simulation models for bus dispatching control and for adaptive signal control, respectively, and then tested the models jointly. The *bus dispatching control model* was developed for evaluating, and eventually implementing strategies along bus routes[6]. Two major bus operation strategies, headway-based control and schedule-based

control, are explored with the model. The headway-based strategy attempts to maintain proper bus headways in order to reduce bus bunching and passenger wait time. The schedule-based strategy enables buses to keep the original schedule, instead of maintaining a desired headway. Buses are controlled to adhere to their own schedule regardless of how much bunching occurs. The *dispatching control model* measures the system performance in terms of passenger in-vehicle time, and wait time, bus travel time and headway regularity. In addition to the above two models, a *signal control model*, a pre-timed phase-based control procedure, was developed. This model evaluates a combined cost function of vehicle delay, number of stops, and on-board passenger delay, and develops a preset steady timing plan that minimizes the cost function. The signal control model is not involved in the scope of our research. Interested readers may reference the paper for details [6].

I-3-2. Previous Work at the University of Ottawa:

The two Systems Science master theses, which preceded our research, constitute the basis of the current work.

In "Study of Real Time Bus Control Using Computer Simulation"[4], Lule Chen examined the application of OPSSIM 1.00 for simulation of the operations of OC Transpo route 95, and performed extensive validation experiments which indicated that the model made reasonable predictions, and that further work using it was justified. In addition, Chen implemented the model on a UNIX platform for easier use. New headway-based bus control strategies are presented and compared with other known control strategies. The statistical analysis and comparisons of those control strategies are based on test results obtained using OPSSIM. Statistical analyses demonstrated that the new headway control policies (called "Message Board" and "hybrid" policies) provide better service reliability in terms of less variable headway for the high-frequency bus route. Finally, the author designed a new control prototype, a knowledge-based system that is a computer program that uses the knowledge and inference procedures of human experts to solve difficult problems. This new prototype system was tested using the simulation model, and will eventually become a building block for a real-time system to assist human controllers.

In her thesis "Study of Dynamic Headway Control Bus Dispatching Rules"[5], Ji Xu focused on the headway control strategy. Previous research shows that the headway control strategy is more appropriate for improving

high frequency bus service than the traditional schedule control strategy. Ji first developed and implemented a customer survey to determine what is the largest headway, i.e., the time between the departure of two successive buses at stops of a given route, so that the headway control strategy is the preferred bus control strategy. Ji observes that traditional headway control policies have been shown to be very effective in keeping the headway constant in systems which are not experiencing "shocks". However, these policies suffer from an inability to adjust quickly to the system shocks, such as bus breakdowns, road blockages, etc. Based on these observations, Ji proposes an improved holding policy, "flexible scheduling", in which buses along a route "auto-regulate" themselves using real-time data obtained through the Automatic Vehicle Location Control (AVLC) system. "Flexible scheduling" for high-frequency routes is achieved by implementation of a dispatching rule used which determines the trip departure times dynamically at terminals and possibly other control points based on the actual achievable headway between successive buses. These policies were tested and verified using OPSSIM. Finally, a user-friendly interface which provides easy access to both the simulation model and a control module was developed to provide a better working environment to the users of these models.

I-3-3. Terminology and concepts related to public urban control strategies:

According to the study in the literature, Levinson classifies control strategies in the urban public transit system into two groups [3]. *Routing and scheduling strategies* deal with medium-term planning, providing stable schedules for periods generally at least a month long. *Real-time control strategies* are designed to remedy specific problems as they occur during the day.

Among the real-time control strategies, two classes of real-time control approaches should be distinguished:

- Schedule adherence: the objective is to keep buses as close as possible to their printed and planned schedules;
- Headway control: the objective is to maintain uniform spacing (with respect to time) between buses on a route.

I-3-3-1. Schedule and Headway Control Review:

As mentioned above, there are two basic approaches for dealing with real-time bus control, namely *schedule* control and *headway* control. The various real-time control actions, e.g., insertion or cancellation of a full or partial trip, etc., are generally applied to achieve either schedule or headway control. We briefly provide details of schedule and headway control below. First, we list the major reasons that cause buses to become off schedule:

- Bus breakdown or accident
- Traffic congestion slowing down busses
- Smooth traffic flow resulting in speeding up of buses
- Response to signal system (are traffic lights green or red?)
- Passenger density at stops
- The speed of the bus
- Passenger boarding and alighting times

The discussion below is broad, and includes discussion of a large range of possible actions which can be simulated. Our simulation work forcuses on one such type of action, namely "inserting a trip".

Schedule adherence control:

This strategy is to push the off-schedule buses (early or late) back to the planned schedule. The off-schedule buses are the buses for which the difference between actual running time and the scheduled time is beyond a certain threshold time interval called the trip schedule adherence tolerance parameter. Schedule control is appropriate for low frequency routes for which customers schedule their arrivals at a stop to coincide with the time at which the bus is scheduled to depart from a stop. The performance indicators for schedule adherence are:

- Statistics describing schedule adherence for each bus, which is the difference between actual and scheduled departure time;
- Statistics describing the percentage of buses that are early, on-time or late at one control stop.

- Statistics regarding waiting time, transit time, etc., for riders.

Note that the statistics mentioned above may be computed over individual or multiple stops, various time intervals (e.g., rush hour, off-hours), etc.

The following control actions have been implemented in various places for schedule adherence control:

- Skipping stops or short-turning when a bus is late: Skipping stops is a control action normally used when a bus runs behind schedule. The bus passes by a stop, if no passenger wishes to alight, even if some passengers wish to board, in order to make up for its lateness. This is normally done on a high frequency route. Short-turning a bus can be used if the bus is very late. It may be desirable to have it terminate before the final stop and make it proceed to its return trip;
- Inserting a temporary bus to replace a disabled bus;
- Canceling one trip when traffic is not as heavy as expected or a bus is running too early.

Headway control:

The objective of this strategy is to space buses to obtain relatively uniform headway between successive buses. Headway control is appropriate for high frequency routes. The performance indicators for headway adherence are:

- statistics describing the headway adherence, which is the difference

between actual and scheduled headway;

- statistics regarding waiting times, transit times, etc., for riders.

Note that the statistics mentioned above may be computed over individual or multiple stops, various time intervals (e.g., rush hour, off-hours), etc.

The following control actions may be used for the control of headway adherence.

- Skipping or short-turning when the actual headway is much longer than planned headway
- Inserting a full trip when the actual headway is much longer than planned headway
- Holding a bus (i.e., delaying departure at a stop) when it becomes to close to its predecessor along a given route
- Inserting a temporary bus to replace a disabled bus
- Leap-frogging when two or more buses are bunching

Our thesis will focus on the design and implementation of bus insertion control actions into the OPSSIM simulation model.

I-4. Investigation in the OC Transpo Control Centre:

As previously stated, part of the work of this research is the development of an on-line user-friendly interface for the central controller, to facilitate real time interventions by the controller. In order to ensure that

design of this interface (display screen) would be as useful as possible to controllers, several visits were made to the control centre at OC Transpo to investigate their control platform, real interface and needs.

In the OC Transpo control centre, there are about ten control operation platforms. Each controller can handle one control operation platform. All central control commands are sent from these control operation platforms to drivers via the telecommunication system. Each controller is responsible for the operation of five to ten routes. All information needed is displayed on the display screen of the desktop on the platform. The screen is designed to be a multi-window display, in which the user (i.e. the controller) can select the window(s) or information bar(s) of current interest by an appropriate mouse click. There are also some text editor boxes which allow controllers to input information into the system. On one particular platform, the screen displays the route 95 map, on which the progress of all route 95 buses can be traced. For example, if one bus arrives at the Campus Station on time, a green light will flash the bus identification number at the Campus Station stop. If the bus arrives after the schedule, a red light will flash the bus identification number. The desktop is also connected to the AVLC. If the controller requires data from one or the other of these two systems, he can print them out from his platform. Moreover, the incoming new data are saved into the two systems automatically.

I-5. Outline of the research:

The work of this thesis includes participation in the design and implementation of an updated OPSSIM model and verification of the modified part in OPSSIM in order to ensure that it is working properly. The work is described, chapter by chapter, below:

Chapter II: Background and Work Plan.

- Comprehending the project background.
- Defining the major mission.
- Making the work plan.

Chapter III: Central Control Actions.

- Clearly pointing out the requirement of Central Control Actions, and the external environment for implementing the Central Control Actions in the OPSSIM simulation model.
- Designing and Implementing the Central Control Actions modules in the OPSSIM simulation model.
- Providing the user guide.

Chapter IV: Run Time Bus Progress Screen.

- Clearly pointing out the requirements for the Run Time Bus Progress Screen, and the external environment for implementing it in the OPSSIM simulation model.

- Designing and implementing the Run Time Bus Progress Screen in the OPSSIM simulation model.
- Providing the user guide.

Chapter V: Validation Tests and Result.

- Designing the test plan.
- Testing the newly implemented code of the OPSSIM simulation model.
- Analyzing the results.

Chapter VI: Summary and Future Research.

- Summarizing the enhanced capabilities of the new version of the OPSSIM simulation model.
- Recommending areas for future research.

II. BACKGROUND AND WORK PLAN

II-1. The background of the project:

The project entitled "Bus Priority System Project" is mandated and funded by Transport Canada and the Regional Municipality of Ottawa-Carleton. TranSys Research Ltd. was hired as part of the project team in October, 1997, and was responsible for developing and implementing the simulation software package that is called OPSSIM. The University of Ottawa team at the Faculty of Administration agreed to work with TranSys

on the design of various real-time control actions for both bus controllers and bus drivers, to program these actions, and to develop and program an on-line control user interface. The steering committee for the "Bus Priority System Project" included: Mr. Brian Marshall (Transport Canada), Mr. Kornel Musci, (Regional Municipality of Ottawa Carleton), Mr. Joel Koffman, (OC Transpo), Professors Francois Julien and Jeffrey Sidney (University of Ottawa), Mr. Gord English and Mr. Tom Moynihan (TranSys Research Ltd.). The steering committee is responsible for the planning, organization, direction, and control of the project. The steering committee's past actions continue to effect the long-term conduct of the project. Under the committee's guidance, the initial work focused upon assessing bus priority performance under downtown conditions (e.g., dedicated-lane one way traffic operation such as on Slater and Albert Street) and under not heavily congested two-way traffic conditions. Major data sources included:

- The Automatic Passenger Counting (APC) System data,
- The Automatic Vehicle Location and Control (AVLC) System data
- Traffic signal operating characteristics and traffic flow data (from Regional Municipality of Ottawa-Carleton, abbreviated as RMOC)
- Intersection layout and bus stop locations (from AutoCAD drawings also from RMOC)

The first three data items above appear to provide an adequate basis for characterizing link delays when used in conjunction with APC and AVLC data.

II-2. The major task of the project:

The Bus Priority System Project consists of seven interrelated parts, namely:

1. Enhance simulation model to include various bus driver control actions,
2. Enhance simulation model to include various bus controller control actions,
3. Enhance statistical output available,
4. Provide an interactive real-time user interface,
5. Provide capability to do multiple route simulation,
6. Provide capability to study system behavior at transfer points,
7. Provide capability to model link delays.

We describe below the initial capabilities of the simulation system OPSSIM as of the start of this research in 1997, the specific goals of the various enhancements, and the implementation of the enhancements. This section is taken from the research proposal prepared by Transys Research Ltd. and is reproduced here because it allows the reader to place the work done in this thesis on Bus Controller Control Actions into the context of the whole project.

The specific elements of the whole project which have been undertaken as part of this thesis have been identified below.

1. Bus Driver Control Actions (not part of thesis)

Initial Capability:

- bus "self regulation" by headway or schedule keeping

Goal of Enhancements:

- improved automated bus driver's control actions
  - * delay
  - * speed-up
  - * bypass stop
  - * leap-frogging of buses under close headway conditions

Implementation Issues:

- implementation may be similar to existing "self regulation" procedures
- leap-frogging should maintain nominal trip order

2. Bus Controller Control Actions(thesis: chapter III and chapter IV)

Initial Capability:

- control action requires change in database information and restart of simulation

Goal of Enhancements:

- provide automated bus controller's control actions for:
  - * insertion of a bus at control points
- add manual control capability for above actions

Implementation Issues:

- automatic/manual control capability will require ability to update bus trip database(chapter III)
- automatic/manual control implementation will require a messaging capability (chapter III)
- manual control capability requires a new controller's interface (chapter III)
- displays simulation progress and performance statistics (chapter IV)
- accesses trip assignments (chapter III)
- automatic control actions will require bus rescheduling
  - internally

3. Output (not part of thesis)

Initial Capability:

- provides output of accumulated performance statistics in a text file

Goal of Enhancements:

- supports reporting of effects of enhanced capabilities
- provide default output specification to ensure ease of use
- facilitate customization by users to suit their particular analysis needs

Implementation Issues:

- text base output versus use of database
- quantity of output and ease of use must be considered

4. User Interface(not part of thesis)

Initial Capability:

- provides menu based program setup (i.e. selection of various scenario files)
- progress of simulation depicted using text based user display

Goal of Enhancements:

- to provide a Graphical User Inferface (GUI) typical of an AVLC system

Implementation Issues:

- data structure must support sufficient route information to allow meaningful display of progress along routes
- coding of efficient C language graphics procedures
- attempt to strike balance between code portability and appropriate use of Windows development environments

5. Multiple Routes(not part of thesis)

Initial Capability:

- simulate simultaneous movement of many buses on a single route only
- not possible to capture delays induced by laneway interaction with buses on other routes
- not possible to capture effects of interaction with buses on different routes sharing stops

Goal of Enhancements:

- facilitate simulation of hub and spoke type route configurations
- allow assessment of consequences of delaying buses with transferring passengers
- routes to be totally independent (i.e. routes sharing laneway will not interact)

Implementation Issues:

- likely not practical to perform an entire network simulation
- stop implementation must facilitate sharing between different routes (new data handling and unload/load procedures)
- inclusion of link delay characterization due to bus congestion (several routes sharing same laneway resources)

6. Transfer Points(not part of thesis)

Initial Capability:

- not supported

Goal of Enhancements:

- permit interaction between routes at one or more user defined stops
- reflect passenger transfers in stop arrival rates
- allow "wait for connection" control actions

Implementation Issues:

- bus arrival rates & passenger transfers must be identified for each transfer stop

7. Link Delay(not part of thesis)

Initial Capability:

- random number process to assign link transit times from a statistical representation
- link transit times supplied for various times of day

Goal of Enhancements:

- inclusion of explicit automobile congestion measures
- inclusion of congestion due to operation of other buses on same laneway

Implementation Issues:

- new data and heuristics required to develop delay relationships

In partial fulfillment of item 2 above, this thesis focuses on the design and implementation of one central controller control action namely the bus insertion, the user interface for the central controller to choose control actions, and the run time screen for displaying the bus progress. In addition, validation experiments for the enhanced OPSSIM simulation model were performed.

Based on the software life cycle, we organized our work into six phases:

- PHASE I: Familiarization with Model and Requirement Investigation
- PHASE II: Requirement Analysis
- PHASE III: System Design and Module design

- PHASE IV: Coding and module test
- PHASE V: Integration and system test
- PHASE VI: Identification of future work requirements

In phase I, in order to become familiar with the exiting OPSSIM simulation model and its operating environment, we ran the program several times under the various scenarios and tested every function provided by the simulation model based on the user manual [1]. At the same time, we reviewed many documents related to the simulation model, such as input data specifications, output report file descriptions, and Lule Chen's thesis: Study of Bus Control Strategies By Computer Simulation[4]. In addition, visits to the OC Transpo control center and interviews with the chief controller and several other controllers enabled us to learn first-hand and understand the pragmatic requirements of a useful interactive system. For example, we were able to establish which information is important for real-time control, which kind of interface would be most useful, which control actions are used frequently, under which kind of conditions these control actions are used, how effective these control actions are perceived to be by the controllers, etc. All of this information is of crucial importance in determining which control actions to implement in the model, and what the interface should look like.

In phase II, we (i) analyzed requirements obtained in phase I under the existing simulation system environment, (ii) decided which requirements were necessary and also feasible for implementation in the existing

simulation system, and (iii) described these requirements in detail for use in the next phase.

Phase III is the core work of the project. We did the system design based on the requirements identified in phase II. The whole project can be separated into three sections. They are: adding bus controller control actions, creating the control interface to enable the bus controller to choose control actions, and displaying the bus progress information on the OPSSIM run time screen. For each part, we did the module design. The analysis of the logical relationships among the existing simulation programs, data files, and newly embedded programs required a significant amount of effort. Some of the standard tools were quite useful: flow charts, module relationship charts and component analysis in particular. Once all of the relationships were sorted out, individual modules were designed.

Phase IV, coding and module testing, involved writing and modifying embedded programs in a "moving target" environment: while our work was progressing, the main simulation model was simultaneously being modified by Transys in Kingston. Thus, linkages and data files had to be dealt with carefully to ensure compatibility with the latest version of OPPSIM. Each module was tested for reliability in isolation.

The major work of Phase V was to finalize the integration of the new and the revised modules with the existing OPSSIM model. After integration, test data sets were chosen and the integrated system was tested, debugged and finalized. The result was the enhanced version of OPSSIM.

As this is a complex system, and a major undertaking, the work proposed in the project far exceeds the scope of this thesis. In Phase VI, we summarize work that has not been done in this thesis in order to provide guidance to whoever would continue this project.

III. CENTRAL CONTROL ACTIONS:

III-1. Purpose of the Development of Central Control Actions:

III-1-1. Comparison between central control actions and driver control actions

The difference between a central and a driver control action is who is responsible for initiation and manipulation of the control action. In the case of a central control action, initiation and manipulation are in the hands of the central controller. Central controllers base their decisions upon on-line information of all controlled buses and roads. Central controllers who have more experience can observe abnormal performance of controlled buses and/or road conditions, e.g., an incapacitated bus or a traffic jam, from the regular bus operation data, and he/she also can anticipate some undesirable events. If the undesirable event really happens, or if they expect that it will happen, the central controllers are empowered to intervene by sending commands to drivers to adjust their actions in order to ameliorate current or anticipated negative situations. For example, if a bus

appears to be overtaking the bus ahead of it, the controller might request the driver of the bus which is ahead to implement corrective actions, such as, skipping a stop if there are no passengers wishing to alight. The system to support these control actions is called the central control system, and is illustrated in the figure below.

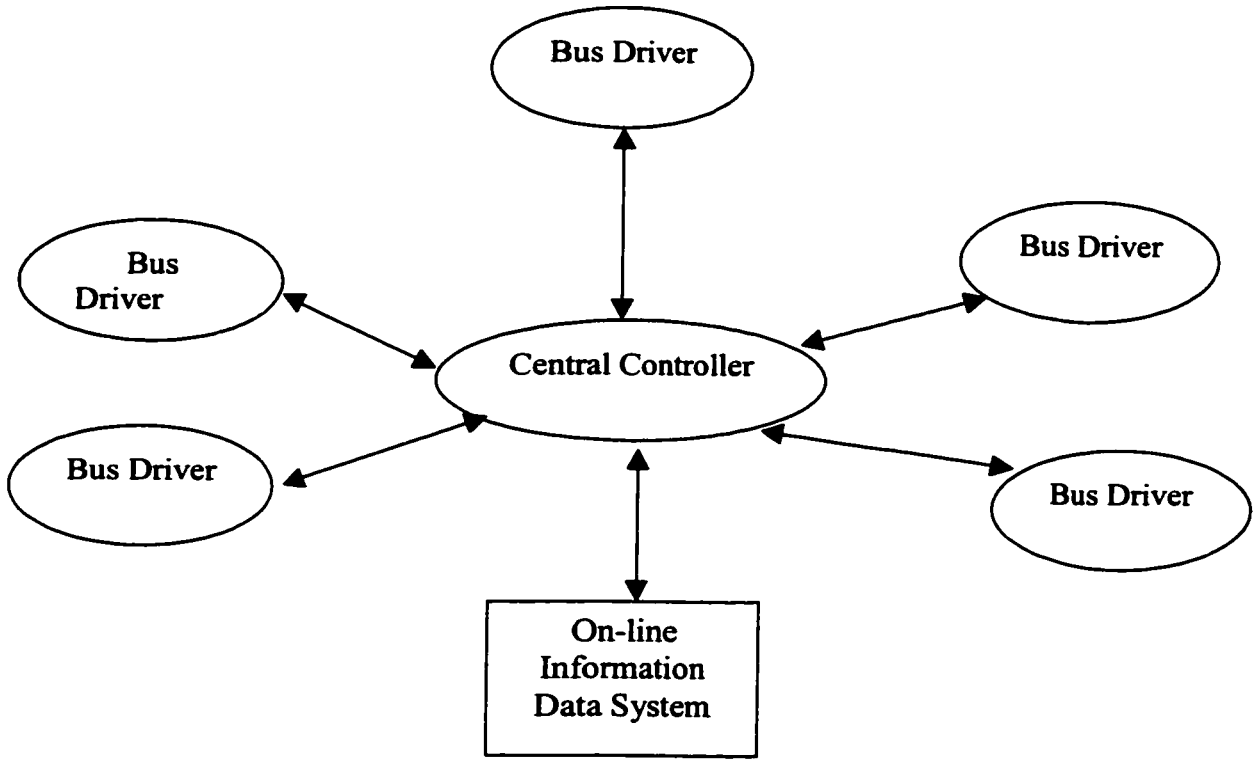


Figure 1: Central control system model

Where:

↔: communication channel.

Figure1 describes a central control system. In this model, the controllers are the core of the system. They gather the required information from each driver and from the on-line information data system through the

communication channel. They are also responsible for analyzing the information, choosing the control action and sending the decision back to the driver through the same communication channel. So, all control decisions are coming from controllers. The driver is only responsible for providing the on-line information and carrying out the control decision, but not participating in making the decision. In this model, the responsibility of making the decision and the responsibility of doing the action are separated.

Some commonly used Central Control Actions are:

- Insertion of a bus at a control point: At a control point, at the defined time, a bus taken from a pool of extra buses in stand-by mode is inserted to run a scheduled or a non-scheduled trip.
- Short turn of a bus at control points: At a control point which is not the destination terminal, at the defined time, the bus interrupts its current trip and turns back to the starting terminal.
- Run-through to the end of the route: At the control point, the bus will drop off all passengers not heading for the destination terminal and run through directly to the destination terminal without stopping at any bus station between the control point and the destination terminal.
- Trip cancel: a whole trip from the starting terminal to the destination terminal or part of a trip from the control point to the destination terminal is canceled.

Driver control actions are initiated by a bus driver. The drivers, not only focus on driving the bus, but also make the travel decision based on the information which is immediately available to them, such as the road condition, and their adherence to the schedule. At present, driver control decisions are normally taken independently from the status of other buses along the route. This could be described as a decentralized, uncoordinated system. In this system, driver control actions are normally taken in isolation of other buses, without knowing the status or position of other buses. These decisions normally are based on schedule adherence.

With the continuous improvement in communication technology, it is possible to envision the emergence of decentralized, coordinated systems, as illustrated in the figure below.

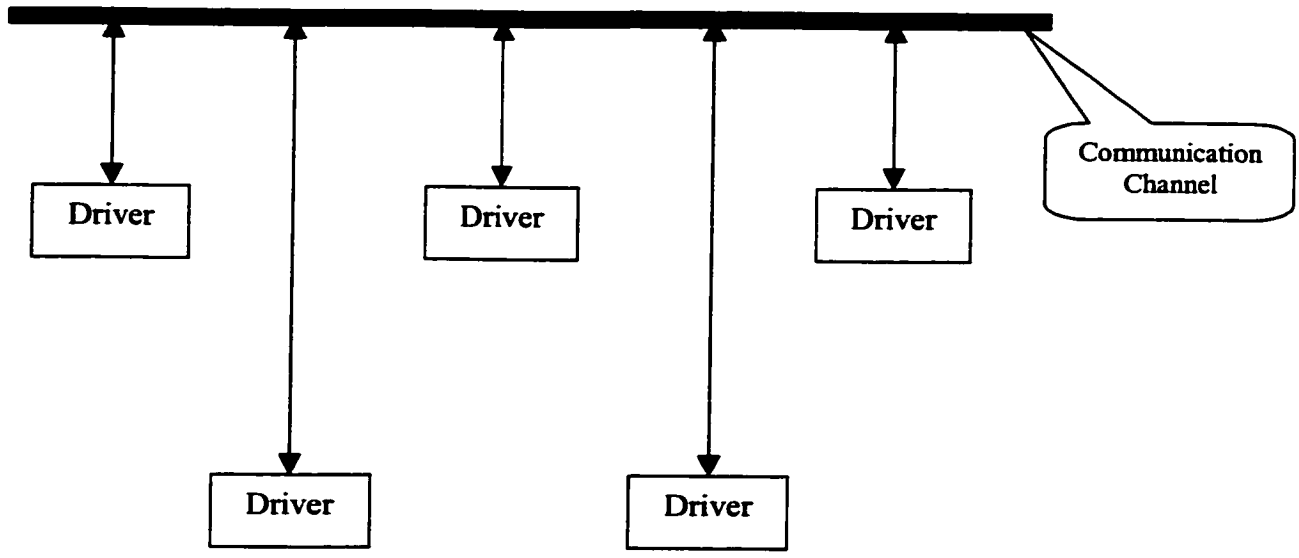


Figure 2: Decentralized and coordinated control system model

Compared with the Figure 1, there is no central controller in the system shown in Figure 2. The authority of making the control decisions, which belongs to the central controller in the central control system model, is moved to the driver in this model.

From the above figure, in a decentralized system, a bus driver would have information about road condition and the current status of other buses such as the headway between his bus and the previous and next bus through the communication channel. The communication channel can be a wireless telecommunication system. Drivers can talk each other to get the required information, such as the location, by mobile phone. This can also be achieved via the e-Information system. One screen is installed on each

bus, which is connected to the distributed information centre. The data required for making the control decision will be shown on the screen to the driver from the distributed information centre. Based on the information, bus drivers would be empowered to take some actions by themselves in order to improve upon the performance or solve the trouble they have met.

Such a system is theoretically possible, if we provide brief, clear and specific information to the driver, and the driver gets certain training. Experienced drivers would probably make more effective decision than inexperienced drivers. But in practice, it is presently very difficult to run this system, because of the cost and also a lack of necessary technology and support system, such as a distributed information system. One would also need to study how to distribute such responsibility to drivers. It could affect their driving duties and then the safety of passengers.

Under a decentralized and uncoordinated system, driver actions are limited to the following:

- Hold the bus at a bus station: if a bus enters the bus station too early with respect to its schedule, or an incoming bus with riders wishing to transfer to the current bus is expected soon (radio communication), the driver may wait at the stop before resuming the trip.
- Slow down: may be implemented by a driver if a bus is running ahead of schedule.
- Speed up: may be implemented by a driver if a bus is running behind schedule.

- Bypass stop: if a bus is very late, and there is no passenger wishing to alight at the station, the driver may bypass a stop, even if there are passengers wishing to board there. This is most likely on a high frequency route such as OC Transpo route 95, where it can be assumed that the passengers waiting to board will be picked up soon by following bus.
- Leap-frogging: the drivers of two consecutive buses on a given high-frequency route that are experiencing a situation of zero headway (i.e. bus bunching), may select to leap-frog to get better service quality. Typically, the buses will stop at alternate stops, and the stopped bus will be passed by the other. This is most feasible during rush hour and on sections of the route on where alighting is rare, but boarding is common.

In our thesis study, we set our core work around the central control system model.

III-1-2. Need for inserting a central control action module into OPSSIM

The main reason for us to develop the central control action model in OPSSIM is to enhance the simulation capabilities of OPSSIM to facilitate a wider range of experimentation.

The OPSSIM simulation model is used as a tool for the testing and evaluation of different service control strategies. It is also a "training

simulator" for current and new service control staff. In its present version, the model can simulate the bus performance for only one single route, and it can also be used to examine the performance of schedule control and headway control strategies. However, there is no interface between the controller and the simulation model to enable the controller to insert central control actions during the run time. Therefore, the present model can't be used to simulate on-line control. For example, if the controller observes that a bus at Campus Station has broken down on an east bound trip during a simulation run, he might want to insert another bus from a nearby garage to finish the run of the broken bus. Prior to our work, the OPPSIM model didn't provide the capability to perform such an intervention.

In order to make significant inroads into bridging this gap between the existing model and a broader, more flexible model, the following tasks were undertaken:

1. Design and development of a user interface to enable users to introduce the central control actions during the simulation run.
2. Design and development of a basic central control action module, allowing users to insert an extra bus following the breakdown of a regular bus.

III-2. Design of Central Control Actions:

III-2-1. External environment of the central control action module:

The central control action module will be an embedded section in the enhanced OPSSIM simulation model. Previous to devising a good design, it is necessary for us to first clarify the external environment of the central control action module.

III-2-1-1. Interface between central control action module and run time OPSSIM simulation model:

During the execution of OPSSIM, the user may want to insert a central control action. The F8 function key on the keyboard will now provide the gateway between the running OPSSIM simulation and the central control action module, as shown in the run time screen of the OPSSIM simulation model (Figure 3). Immediately after F8 is pressed, the central control action module is activated and the OPSSIM simulation run is paused. The main menu for the central control action module appears, and, the user can send the central control action command into the OPSSIM simulation after following menu-driven instructions.

The F8 function key is the only interface allowing the user to interrupt a simulation run to insert a central control action.

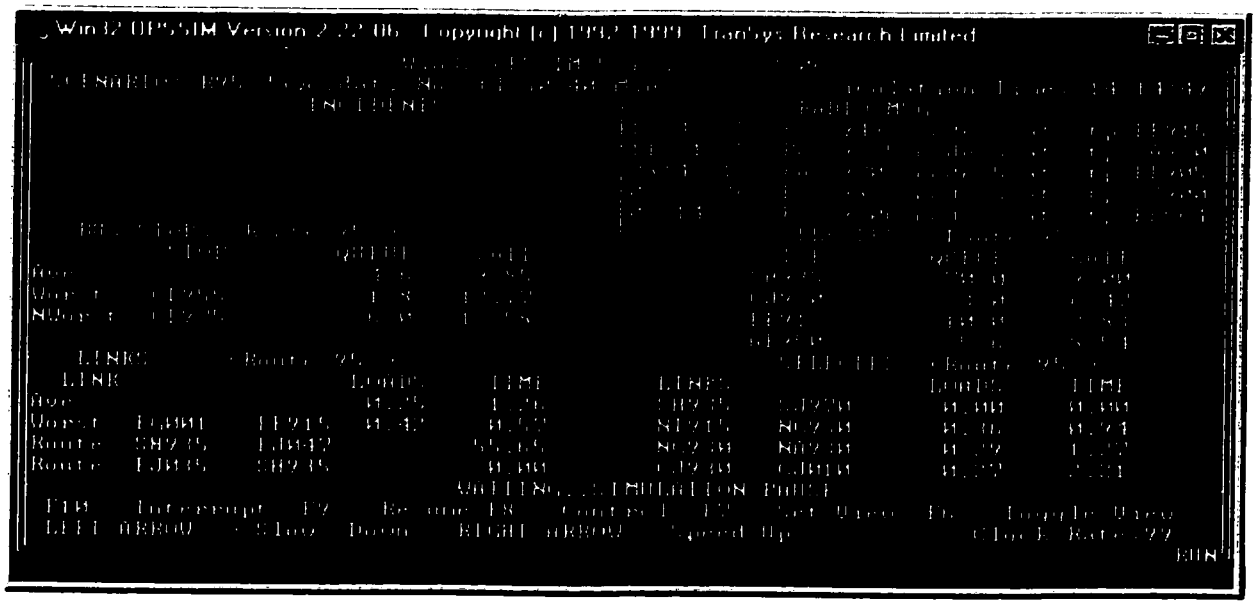


Figure 3: Run time screen of OPSSIM simulation model

III-2-1-2. Input for the central control actions module:

III-2-1-2-1. AVLC data files:

Four AVLC data files constitute the whole input data source to the central control action module. They are Trip.dat, Block.dat, Trippat.dat, and Trippatvc.dat. Each data file has its own format and function. Before explaining each data file, we should first introduce some basic terminology associated with the AVLC data.

A stop is a specific location, such as a street corner or a shopping mall for example, where passengers can board buses, or alight from buses.

In the development of schedules, the planned departure times of buses from only a sub-set of all existing stops, known as "nodes" or "control stops" or "time points", are published. Nodes are typically major stops where

a number of bus routes converge. At OC Transpo, nodes are normally equipped with an AVLC detector, allowing controllers to monitor bus progress.

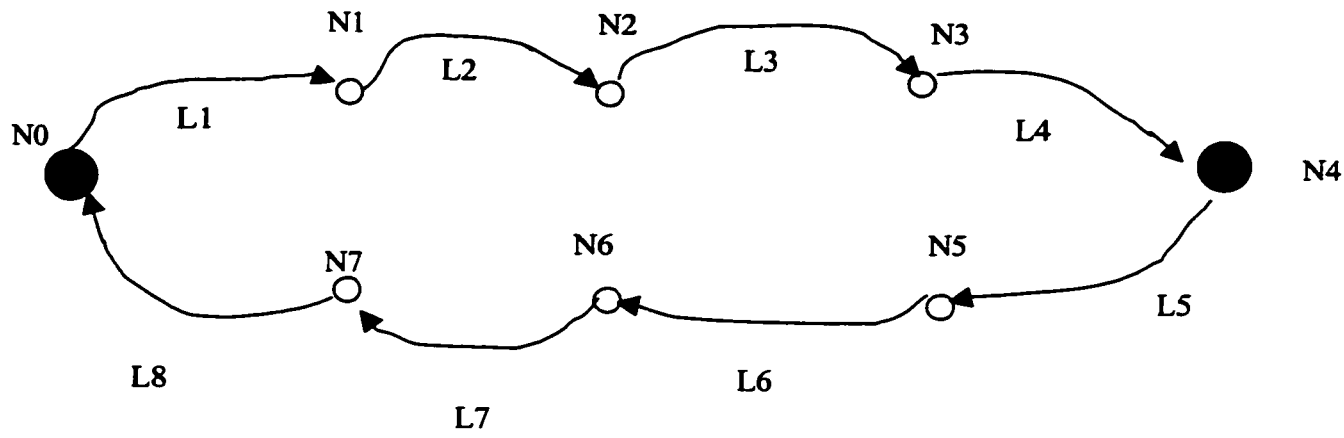
A link represents a path between two nodes. There may be intermediate stops along a link.

A bus route identifies all stops to be visited by buses. A route is defined by one or more route patterns. A route pattern, also known as a route map, is composed of links, and defines the sequence of stops to be visited by buses in a given direction from a start terminal to a destination terminal. In some cases, the start and destination terminal is the same stop. A terminal stop is a node.

Buses operating on a given route will complete one or several trips along the route. A trip is a path comprising one or several links between two nodes. The start and destination nodes of a trip are normally terminals of the route, but trips may start or end at other intermediate control stops. A trip is defined by its start time, and by its trip pattern. A trip pattern is associated with a route pattern, and specifies the planned running times over all links of the route pattern.

Finally, a block is a collection of consecutive trips on a given route which are assigned to a specific bus. The block starts from the time at which the bus begins its first trip over a given route, to the time at which the bus completes its last scheduled trip over that route. A bus may be assigned several blocks (i.e. may be operating over several routes) on a given day.

The following diagram illustrates the terminology defined above.



Where:

● : a bus terminal, also called a node.

○ : a bus intermediate control stop, also called a node.

→ : a link.

All ● + all ○ + all → a route.

Trip	Trip Start Time	Link	Link Travel Time	Link	Link Travel Time	Link	Link Travel Time	Link	Link Travel Time	Trip End Time
1	8:00	L1	15 m	L2	20 m	L3	20 m	L4	10 m	9:05
2	9:10	L5	11 m	L6	18 m	L7	21 m	L8	14 m	10:14
3	10:20	L1	16 m	L2	20 m	L3	18 m	L4	12 m	11:26

Figure 4: Diagram for node, link and trip

Note:

- L1 to L8 represents links along a given route.
- L1, L2, L3 and L4 represent route pattern1, while L5, L6, L7 and L8 represent route pattern 2.
- Trip1 and trip3 share the same route pattern, or sequence of stops; but, trip1 is distinct from trip3 because its start and end times are earlier than those of trip3. Trip1 and trip3 also follow different trip patterns, because the link travel times may differ by time of day due to anticipated traffic conditions (peak hour, for example).
- Trip1, Trip2, and Trip3 form block1, because all these trips are associated to the same route, and also a given bus will run all of trip1, trip2, and trip3.

Next, we will show the schema of each data file. These data files come from the AVLC database. They all have a defined file structure.

Trip.dat file:

The Trip.dat file holds all the information about trips. The schema of the file is in the following table.

Field Name	Data Type	Content	Example	Notes
iLineId	integer	Includes the information about the route number, trip type, and the trip direction.	95010	95: the route Id (route 95). 01: the trip type (this trip is running in the week day). 0: the trip direction (east bound).
iStartTime	integer	The start time of the trip	934	Unit: minutes
iTripId	integer	The identifier of the trip	934095010	Unique value in the file. It is the primary key to find the trip record. iStartTime + iLineId
iTripPatternId	integer	The identifier of the trip pattern	9507	
iBlockId	integer	The identifier of the block	95030	
nBlockSeq	integer	The sequence number of the trip in the block	1	1: The first trip in the block.

Table 2: The schema of the trip.dat file

Trippat.dat file:

The trippat.dat file holds all information about trip patterns. The schema of the file is in the following table.

Field Name	Data Type	Content	Example	Notes
iTripPatternId	integer	The identifier of the trip pattern.	9512	9512: the 12th trip pattern for route 95
iRoutePatternId	integer	The identifier of the route pattern.	9520	9520: the 20th route pattern for route 95
nDir	integer	The direction of the trip.	0 or 1	0: east bound 1: west bound
nRouteId	integer	The identifier of the route.	95	
nSeqNo	integer	The sequence number of the trip pattern in the specific route under the same direction.	12	12: the 12th trip pattern for route 95 in east bound
iStartNode	integer	The start node id of the trip pattern.	7010	
iEndNode	integer	The end node id of the trip pattern.	7100	
iTripTime	integer	The scheduled total running time of the trip under the trip pattern.	38	Unit: minutes
iDistance	integer	The distance of the whole trip under the trip pattern.	21500	Unit: meter.

Table 3: The schema of the trippat.dat file

Trippatvc.dat file:

The trippatvc.dat file holds all information about trip pattern vector. The schema of the file is in the following table.

Field Name	Data Type	Content	Example	Notes
iTripPatternId	integer	The identifier of the trip pattern.	9512	
iDeltaTime	integer	The time interval between the start and end nodes of a link.	26	The unit is in minutes
iSegVecId	integer	The start nodeId and end nodeId in a link.	70907080	7090: the start node id of the link. 7080: the end node id of the link.

Table 4: The schema of the trippatvc.dat file

Block.dat file:

The block.dat file holds all information about a block. The schema of the file is in the following table.

Field Name	Data Type	Content	Example	Notes
iBlockId	integer	The identifier of the block.	95030	
nFictionRoute	integer	The route id.	95	
nRunId	integer	The sequence number of the block in a route including the day type.	30	30: the 30th bus running route 95 in the work day
iStartGarageNode	integer	The node id of the start node in the block.	9005	
iSchStartTime	integer	The scheduled start time of a block.	920	Unit: minutes
iEndGarageNode	integer	The node id of the end node in the block.	9005	
iSchEndTime	integer	The scheduled end time of a block.	1124	Unit: minutes
nBusId	integer	The bus id which is running the block	8302	

Table 5: The schema of the block.dat file

III-2-1-2-2. The OPSSIM Library:

The OPSSIM simulation model is a large and complex system. The central control action module will be one part of this model. The model was created in 1992. Since that time, modifications have been made to the first version of the model. Thus, a lot of work to generate the model has been done before we started to create the central control action module. All previous and current work is compiled in the OPSSIM library. When we generate the new OPSSIM simulation model including the central control action module, we use this library. This library gives us the benefit to use the build-in functions and global variables it contains. In order to use the existing code and integrate the central control action module with the OPSSIM model later, we must know about, and understand certain header files which are special files used in the C/C++ computer language. There are six header files. All useful global variables, constants, and function prototypes are declared in them. These header files are:

External.h file: Definition of all imported global variables except the main module of OPSSIM are saved in this file.

Forward.h file: All function prototypes for the OPPSIM simulation model are defined in this file.

Global.h file: All global constants for the OPSSIM simulation model are defined in this file.

Machine.h file: All machine specific constants for the OPSSIM simulation model are defined in this file. Some constants are used to build the user interface.

Struct.h file: All global definitions of C language structures used by the OPSSIM simulation model are in this file.

Variable.h file: All global variables for the OPSSIM simulation model are defined in this file.

These six header files were included in the program of the central control action module so as to enable use of the defined variables, structures, constants, and functions in the library directly. This makes it very convenient for us to communicate with the OPSSIM source code. Because of the copyright of the OPSSIM simulation model, we can't attach these files with this thesis. The reader interested in these files, or those data files mentioned in section III-2-1-2-1, can contact Transys Research Ltd.

III-2-1-2-3. The required external parameters:

In addition to the AVLC data files and the OPSSIM library, some external input parameters are also required to run the central control action model. These parameters will be provided directly by the user via the user interface. Each central control action has its own specific parameters. These will be explained later in this chapter.

III-2-1-3. Output of the central control action module:

The output of the central control action module is less complex than the input of this module. It only consists of four new updated AVLC data files. They are trip1.dat file, trippat1.dat file, trippatvc1.dat file and block1.dat file. The structure of these file remain the same as that of their respective parents (namely trip.dat, trippat.dat, trippatvc.dat, and block.dat, which were described earlier in section III-2-1-2-1); however, some data in each file will be changed to reflect the control action taken by the user. When resuming the pending OPSSIM simulation run, these four new AVLC data files are loaded automatically so that the control action taken by the user is carried out immediately.

III-2-1-4. The software environment:

The former versions of the OPSSIM simulation model were implemented in the C language under the MS-DOS operating system. The central control action module will be embedded into the existing simulation model. In order to facilitate the merging of the two parts, the implementation of the central control action module will keep on using the same language and operating system.

III-2-2. The architecture of the OPSSIM simulation model and the central control action module:

In order to clearly explain the relationship between the OPSSIM simulation model and the central control action module, we will show the two architectures of the OPSSIM simulation model separately. One includes the central control actions module, and the other does not.

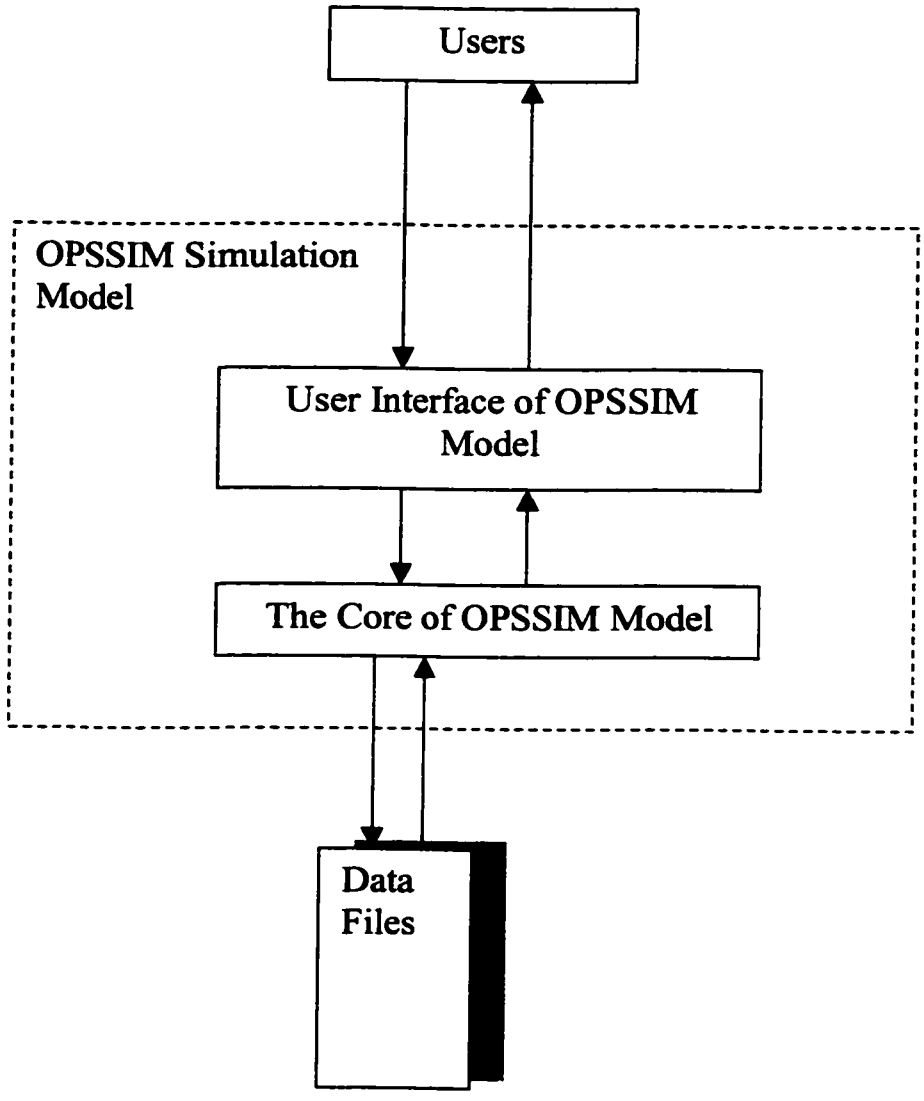


Figure 5: Architecture of OPSSIM simulation model without the central control action module

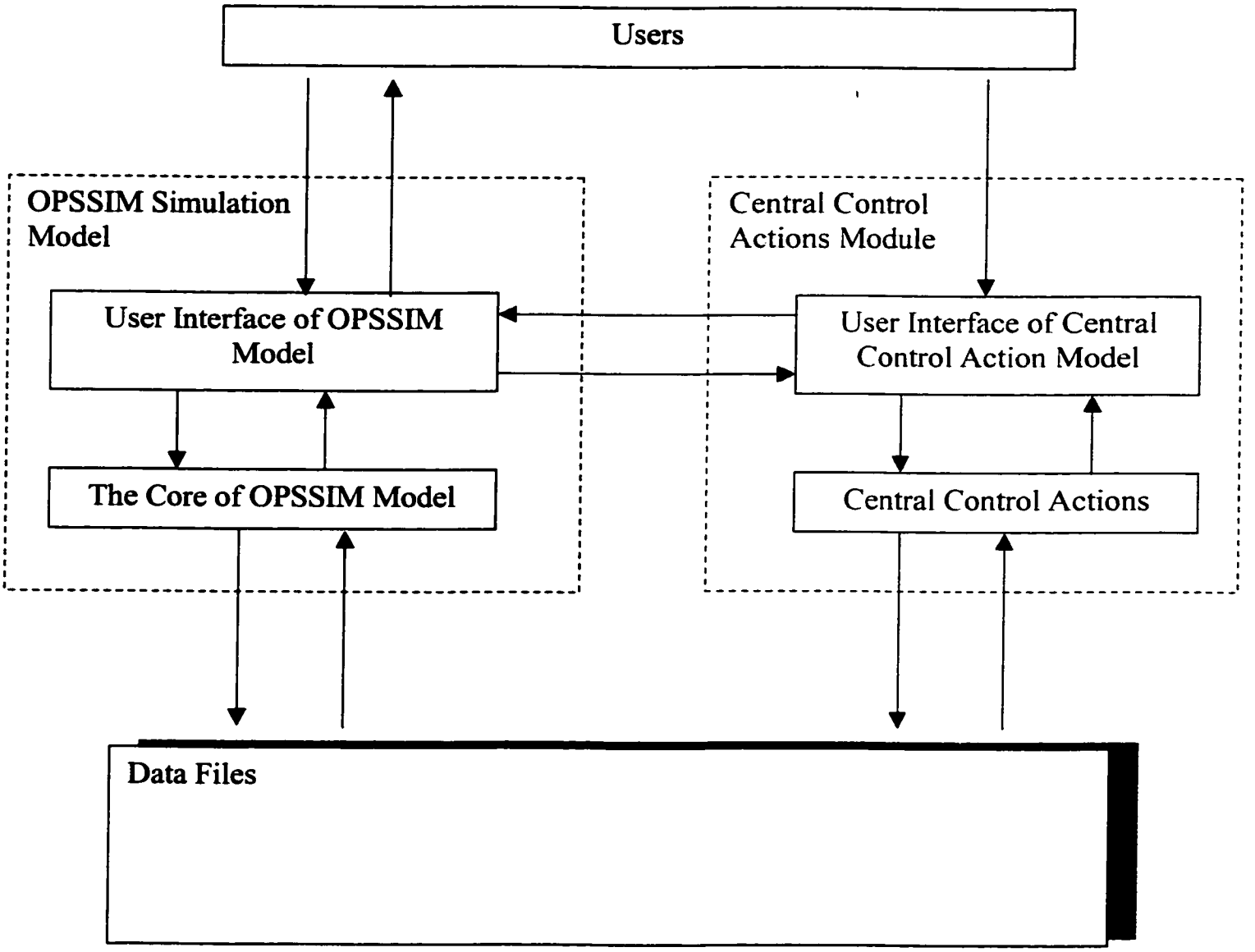


Figure 6: Architecture of OPSSIM simulation model with the central control action module

Figure 6 shows the relationship between the old OPSSIM simulation model and the central control action module. There are two links that connect these two parts together. One is through the User Interface, and the other is through the common data files.

In the Central Control Action module, there are two components. One is the User Interface, and the other is the Central Control Actions. These will be described in the following sections.

III-2-3. The user interface:

The user interface of the central control action module consists of several menus. From the Main Menu window, a number of submenus allow the user to access central control actions, or to resume the OPSSIM simulation run. The central control action submenus are: Insert bus following breakdown, Insert trip, Cancel trip, and Short turn a bus. Selecting one of these control actions leads to a series of submenus specific to that control action. Figure 7 illustrates the Main Menu Window. The various central control actions will be described in detail in the following sections.

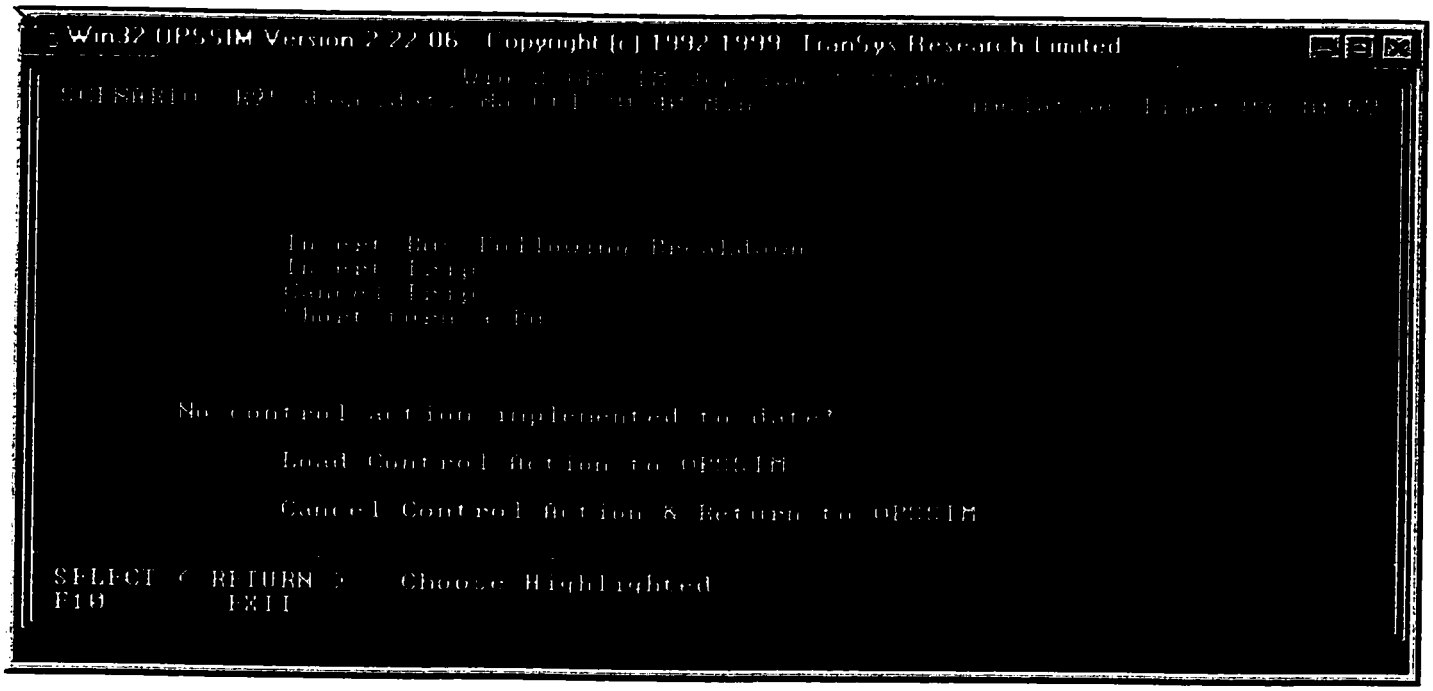


Figure 7: Main Menu

Main Menu:

The Main Menu is the link between the running OPSSIM simulation model and central control action module. The F8 function key is set up as a single directional switch to access the central control action module from the running OPSSIM simulation model. Thus, after pressing the F8 key, the Main Menu appears. There are currently six items on the Main Menu. They are called Insert Bus Following Breakdown, Insert Trip, Cancel Trip, Short-Turn a Bus, Load Control Action to OPSSIM, and Cancel Control Action and

Return to OPSSIM (in Figure7). There is one line used to display messages to the user between the first four items and the last two items: see Figure 7, where the message "No control action implemented to date" appears when the user first accesses this page. Thus, in the future, more actions can easily be added into this module. By selecting "Insert Bus Following Breakdown", the user will open the Insert Bus Following Breakdown Menu, which is described later in this Chapter. The following three items, "Insert Trip", "Cancel Trip" and "Short-turn a bus", are inoperative as the development of these control actions exceeds the scope of this thesis. If the user chooses "Load Control Action to OPSSIM", the pending OPSSIM model will resume and the new data files containing the control action will be loaded. The "Cancel Control Action and Return to OPSSIM" is used to resume the running of the OPSSIM model, without loading any new data files (i.e. no control action is implemented): the same result is obtained by pressing the F10 key.

III-2-4. The central control actions:

In practice, the central control actions which are traditionally used by central controller are Insert Trip or Bus, Cancel Trip, and Short turn. These actions may be used for headway control, schedule control, or to control bus performance. These control actions can efficiently and effectively solve some

common problems which may appear in daily bus movement, such as bus breakdowns or headway fluctuations.

In this section, we will analyze the logic of each control action, and present the methodology and algorithm of how to implement them. Only the "Insert bus following breakdown" action was actually coded and integrated to OPSSIM simulation model as part of the thesis.

III-2-4-1. The Insert Bus Following Breakdown Action:

Normally, buses run several trips back and forth from one terminal to the other. If a bus breaks down at a stop during this loop, what happens? The broken bus cannot continue to run the rest of the trips in this loop. This will result in longer waiting times for passengers, and disrupt the flow of other buses along the route. To see this, assume buses are sequentially numbered as b_1, b_2, b_3, ..., b_m (b_{i+1} follows b_i, i>=1), and stops are represented by s_1, s_2, s_3, ..., s_n (s_1: the start terminal, s_n: the end terminal). Assume bus b_i breaks down at stop s_j. Passengers in bus b_i will alight at stop s_j. When bus b_{i+1} arrives at stop s_j, it will pick up the regular passengers from that stop and the passengers from bus b_i. Thus, bus b_{i+1} will spend more time at stop s_j to board passengers than if no breakdown had occurred. So, bus b_{i+1} will leave stop s_i later than under normal circumstances. Since b_i is no longer operative, there will be more passengers waiting for bus b_{i+1} in the following stop (s_{j+1}, s_{j+2}, and so on). Thus, bus b_{i+1} will spend more time boarding passengers at the following stops. As bus b_{i+1} falls behind its schedule, bus b_{i+2} will likely find fewer

passengers at each stop and catch up to b_{i+1}, with the potential of creating yet another gap between itself and b_{i+3}.

In order to prevent such a scenario from happening, controllers may choose to restore service by inserting an extra bus to which the remaining trips of the broken bus would be assigned. In the above example, the action would introduce an available extra bus downstream from s_j, the stop where the breakdown occurred. The inserted bus will run the rest of the trips of the broken bus.

Methodology:

There are two major issues that need to be discussed here. One is the bus entry point along the route, and the other is the availability of the extra bus.

The Insertion Location of The New Bus :

There are many stops on the route pattern. Which stop is the feasible stop for the new bus to enter the system instead of the broken bus? Most people will choose the stop where the bus has broken down, but this option is not feasible either in real life or in the OPSSIM simulation model. OC Transpo route 95, which is simulated by the OPSSIM model, is a high frequency route. The regular headway of this route is a short 5 minutes. When a bus breaks down at a stop, the driver needs time to report the situation to the controller, and the controller also needs time to find an available bus to insert. Once an available bus is found, it also takes time for the new bus to reach the stop where the breakdown occurred. Hence, it will

normally take more than 5 minutes, the planned headway, to insert a new bus, which means controllers will normally opt to introduce a new bus downstream from where the breakdown occurred, at a point where the extra bus can resume the trips left by the broken bus on time according to the schedule.

Hence, feasible insertion locations for the new bus to enter the system are stops which are downstream from the breakdown location on the current trip which the broken bus was completing at the time of breakdown, as well as all nodes on the following trip in the broken bus's schedule. As exact information regarding the scheduled departure times only exists for control stops (both in the simulation database and in the real system), we limit the feasible insertion of a relief bus to control stops (or nodes). The system will ask the driver to provide the ID of the bus which broke down. This allows the system to propose to the user four control stops, located downstream from the breakdown, as possible insertion points, as well as the insertion time for each location as specified by the schedule. The user can consider the different options and choose one. If there is no control stop downstream from the breakdown location and no more trips remaining in the schedule of the broken bus, the system sends a message to the user and no additional bus is inserted into the system.

The Bus Pool For The Extra Buses:

We design a bus pool in the Insert Bus Following Breakdown action module to model garages where extra available (stand-by) buses are located

in the real system. If the bus pool is empty, it means that there is no available bus. When a user tries to insert a new bus into the system, the program will first search for an available bus in the bus pool. If there is an available bus, the program will pick up the bus and remove it from the bus pool; otherwise, it will send a message to inform the user that there is no available bus, and that bus insertion is not possible.

Algorithm:

The Insert Bus Following Breakdown module starts by searching for an available bus in the bus pool. If no bus remains in the bus pool, it will send a message to the user and return to the Main Menu (Figure7); otherwise, it will remove the bus from the bus pool, and get the bus identifier of the broken bus from the user through a menu driven user interface. This bus ID appeared on the message board in the OPSSIM run time screen (Figure 3) when the breakdown occurred. Then, the system searches which trip the broken bus was running based on the current simulation time. After finding the specific trip, the system continues to search locations where the new bus could be inserted into the system. If no feasible location can be found, it will send a message to the user and return to the Main Menu(Figure7). Otherwise, it will provide a list of up to four locations with proposed insertion time to the user. The insertion time is the scheduled arrival time to the location, according to the trip pattern. After the user chooses a location, the system will send the bus to the location. The new inserted bus will run all trips which should have been run by the

broken bus from that location, and that time. After the last trip is run, the inserted bus will leave the system. All data relative to the new inserted bus will be saved in the new data files. When resuming the OPSSIM simulation model, the new data file will be loaded into the simulation system.

The flow chart based on the above algorithm is shown below:

Note:

CCAMi: denotes the test case number in the test plan [see section V-2-1]. In here, the CCAMi is the i^{th} test case in Central Control Action Module test plan. The detailed descriptions for the test plans are in Chapter V.

The interested reader will find in Appendix A the list of variables within OPSSIM which are used and/or modified when the Insert Bus Following Breakdown module is invoked.

Flow Chart:

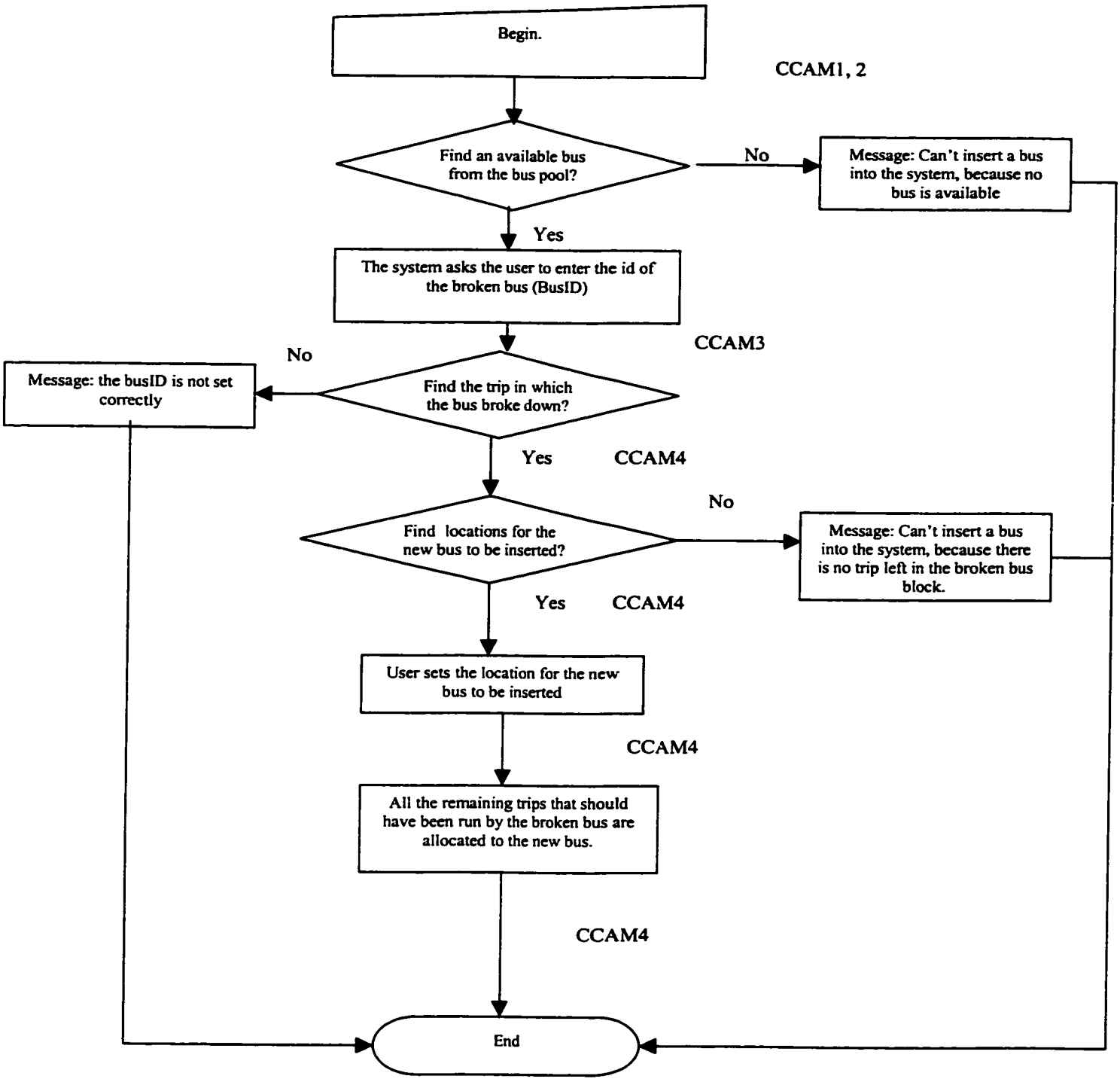


Figure 8: Flow chart for Insert Bus Following Breakdown

Insert Bus Following Breakdown Menu:

On the Insert Bus Following Breakdown Menu, there are two items: Set the broken bus ID, and Run insert bus following breakdown action (Figure 9). This menu is designed for the user to set the parameter required to run the insert bus following breakdown action, and activate this action.

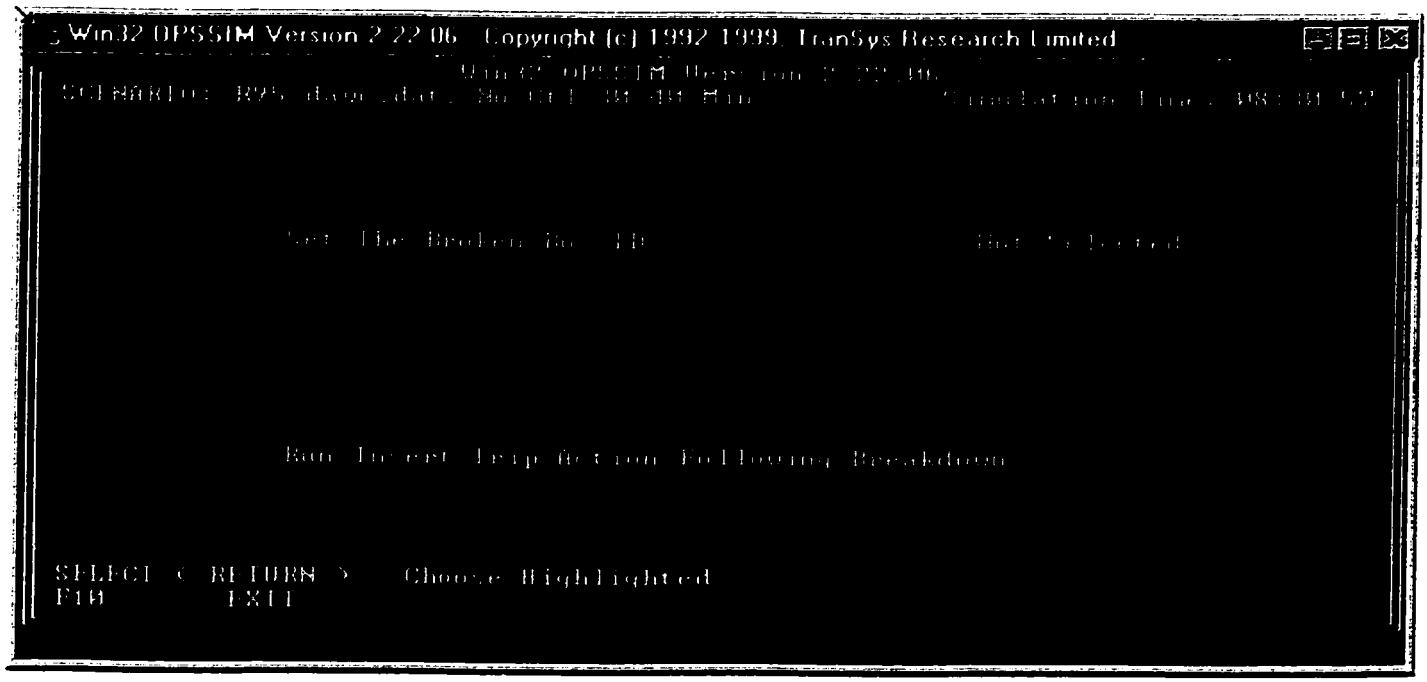


Figure 9: Insert Trip for Bus Broken Menu

After the user correctly sets the required parameter, the insert bus following breakdown action can be run. If there is an error appearing in the insert bus following breakdown action, the error message will show up in the message line in the Main Menu Window. Then, the user can either go

back the OPSSIM without implementing any control action, or reenter the insert bus following breakdown menu.

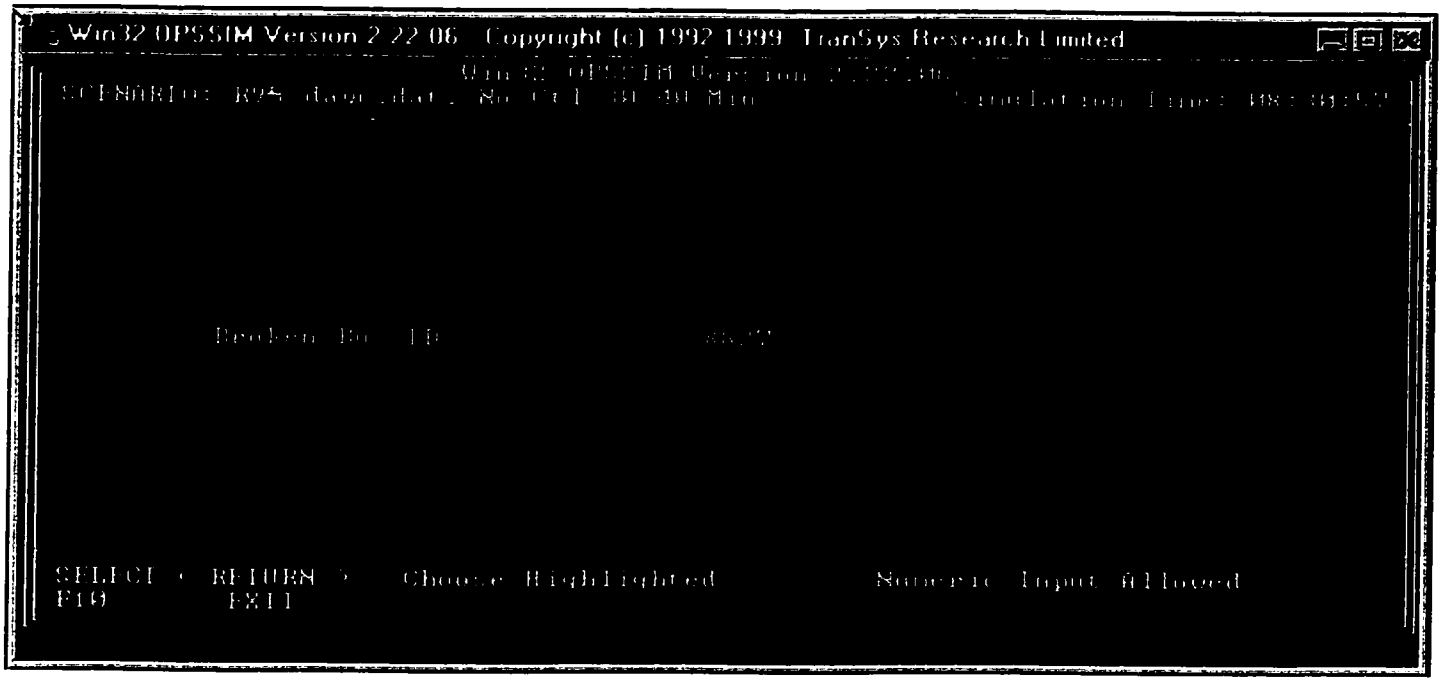


Figure 10: Window for entering the broken bus Id

III-2-4-2. The Insert Trip Action:

The insert trip action module allows users to insert trips for purposes other than bus breakdown. For example, under severe weather conditions, such as a snow storm or freezing rain, buses will typically run late as the number of people taking the bus is greater than usual. Under this scenario, the controller add trips to alleviate some of the pressures on the system.

The inserted trip can be a full trip, from one bus terminal to the other, or a part trip, which just runs a subsection of a full trip. This program is also designed to allow users to add multiple trips. In this case, the combination of these trips can be two or more full trips or a part trip followed by one or many full trips.

Methodology:

The design of the insert trip action differs from that of the insert bus following breakdown action. In the latter, a new bus is inserted and the planned trips that were scheduled to be run by the broken bus are assigned to the new bus, so that there is no need to define new trips. In the insert trip action, either a new bus or an existing bus will run a new trip which didn't exist in the system before. Thus, we must discuss the trip definition and bus considerations below.

Trip Definition:

In defining a trip, we need the start time, a start point and an end point for the trip. The program will be designed to provide a series of windows to allow the user to set these parameters.

At least one new trip needs to be defined in order to successfully insert one or more trip or trips to the system. If users want to insert multiple trips, they need to define each trip one by one through a series of input windows.

The start point and end point of the trip can be the bus terminal, and can also be any control point on the route pattern. The start time of the trip must be later than the time at which the OPSSIM simulation model was

interrupted. This means that it is not allowed to insert a trip whose start time is earlier than the time at which the insert trip command was sent.

Once these parameters are entered, the system will propose one or more trip patterns to the user. The user will choose among existing trip patterns, but will not be allowed to modify these, or create new ones. If a partial trip is inserted, the system will use a sub-section of the selected trip pattern consistent with the selected start and end node to create the partial trip.

Bus Considerations:

In the insert bus following breakdown action, the new bus to be inserted comes from a pool of extra buses on stand-by mode. For the insert trip action, new inserted trips may be assigned to an existing bus currently in operation, or to a new bus from the extra bus pool. Which resource to use will be based on which purpose is served by inserting the new trip. If the user wants to append a trip or many trips to the block of an existing bus, the system will allow the user to choose the bus from the list of existing buses. The system will also allow the user to assign these new trips to a new bus from the extra bus pool, if any is available. This bus pool was described in the previous section on the insert bus following breakdown action.

Algorithm:

The user should first define each trip which will be added into the system, and then search for an available bus. There are two bus resources provided to the user.

If the user wants to get the bus from the extra bus pool, the program will search for an available bus from the bus pool. If there is no available bus in the bus pool, it will send a message to the user: "there is no available bus in the bus pool". Three options are then made available to the user: using an existing bus to run the new trip, redefining the trip or going back to the Main Menu. Otherwise, if an extra bus is available, it will be assigned the new trip, and it will be inserted in the system.

If the user wants to assign new trips to an existing bus in the system, the program will first check whether there is any available existing bus which can take on the new trips, given the block of trips it must already perform according to the schedule. The program will provide a list of all available existing buses, which can take on the new trips, to the user. The user must choose a bus from the list. The program then appends the new trips to the schedule of that bus. If no existing bus can take on the new trips, the program will send a message to the user: " there is no available existing bus", and provide three options to the user: using an extra bus, redefining the trip or going back to the Main Menu.

The flow chart based on the above algorithm is shown below:

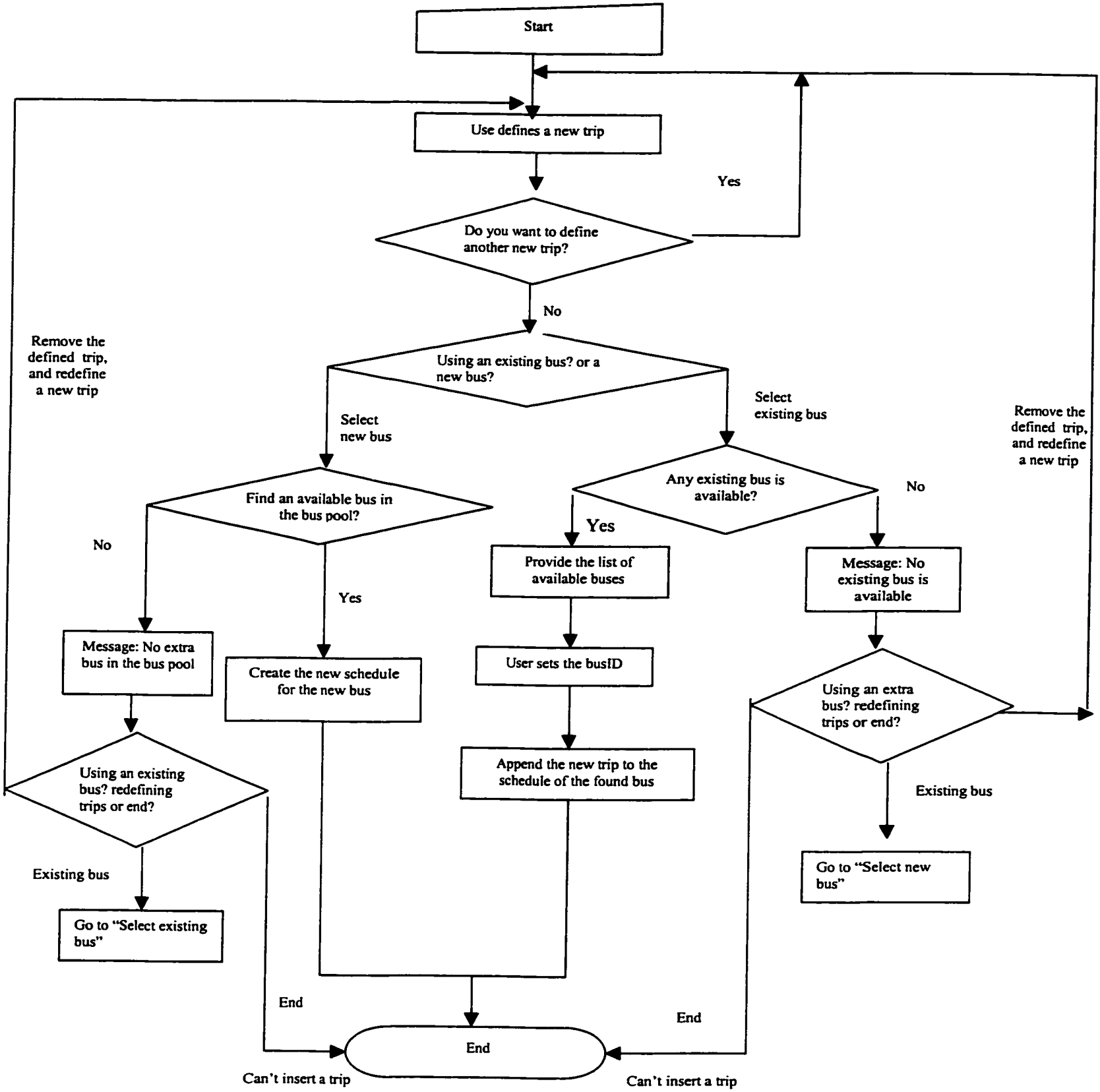


Figure 11: Flow chart for Insert Trip

III-2-4-3. The Cancel Trip Action:

When the actual headway between two buses is too short or when the number of passengers along the route is very low, or a bus on one route is requested on an emergency basis on another route, the central controller usually does the cancel trip action.

In our model, Cancel Trip Action allows the user to cancel full trip, or part of a trip.

Methodology:

Cancel Full Trips:

Two important steps in implementing this action follow:

- Finding the bus that is due to run the trips that are to be cancelled.
- Finding the trip that will be cancelled.

The user will first input the busID. Given the busID, the program will identify all trips which are assigned to this bus, and provide the list of all trips to the user. The user will choose one or many trips to be cancelled from the list and also mark each trip as "Cancelled Full Trip".

Cancel Part of Trip:

In addition to the two important steps described above for the cancel full trips action, the user must also set the cancel point in the trip that will be partly cancelled. When the user chooses the trip to be partly cancelled from the trip list, he will mark it as "Cancelled Part of Trip". When the

program does the cancel trip action for each trip, it first checks whether it is "cancelled Full Trip" or "Cancelled Part of Trip". If the trip has the mark of "Cancelled Part of Trip", the program will provide a list of all control stops on this trip to the user who will select one control stop as the cancel point. The section of the trip from this cancel point to the destination terminal will be cancelled.

Algorithm:

The user first identifies the bus which is supposed to run the trip to be cancelled. If the bus is found, the program will look up all trips that the bus is scheduled to run, and provide the full list to the user. The program allows the user to choose one or many trips from the list and mark the selected trips as "Cancelled Full Trip" or "Cancelled Part of Trip". When the program does the cancel trip action, it will first check the mark of each selected trip. If the mark is "Cancelled Full Trip", the program will directly cancel this trip. If the mark is "Cancelled Part of Trip", the program will list all control stops along this trip to the user. The users will then select a control stop as the cancel point. Then, the program will cancel the part of trip from the cancel point to the end of the trip. After canceling a trip, the program will check whether there is another selected trip that needs to be cancelled and if so, the program will redo the cancel trip action until there are no more trips to cancel.

The flow chart based on the above algorithm is shown below:

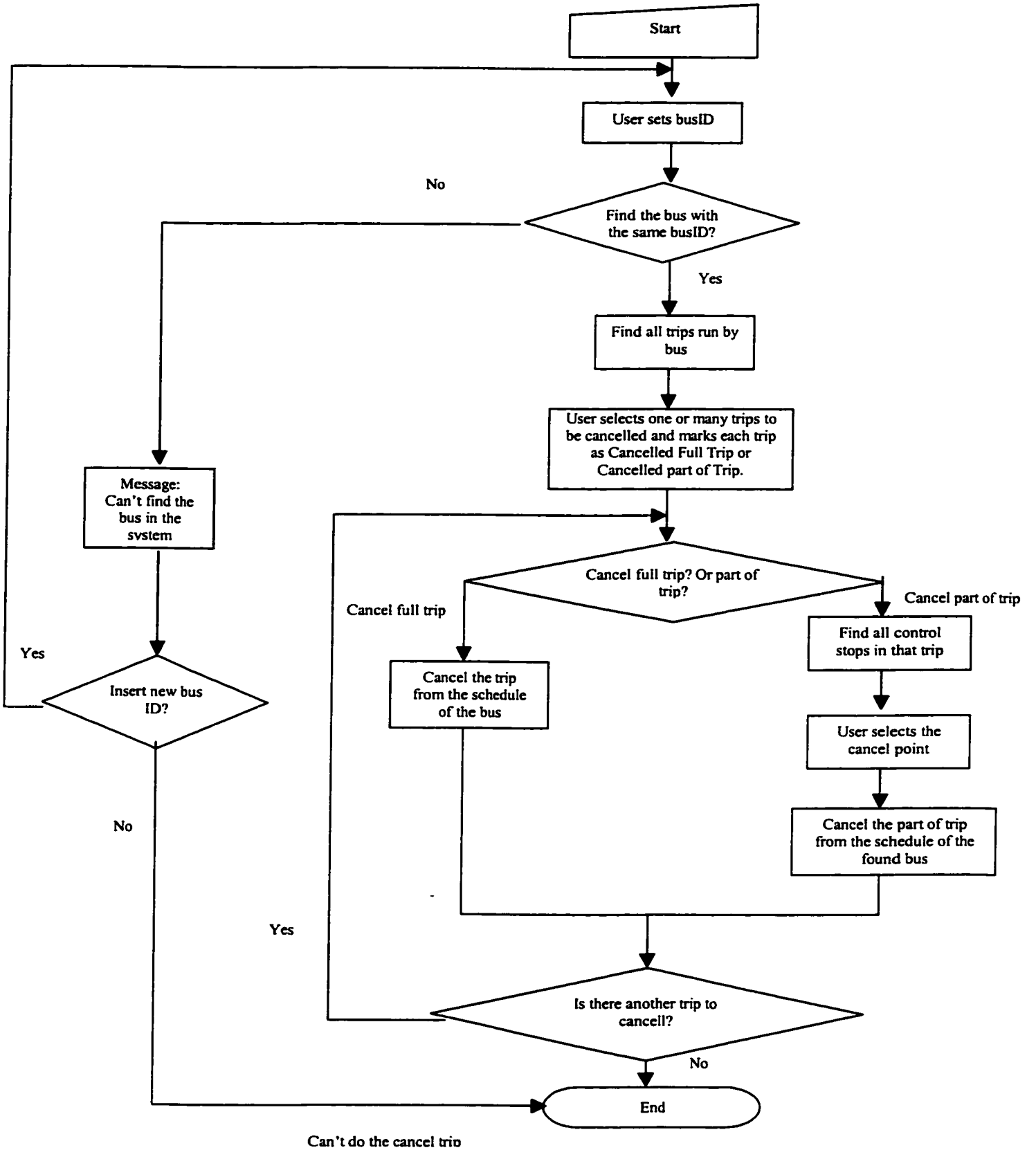


Figure 12: Flow chart for Cancel Trip

III-2-4-4. The Short Turn a Bus Action:

The short turn a bus action may be viewed as a combination of two control actions: a cancel trip action followed by an insert a new trip action. To see this, note that in a short turn, a bus is asked to cancel part of a trip from a specific control stop, the "short turn point", and to start a new return trip from that point. Short turning is used when a bus was significantly behind schedule, in an attempt to bring him back on schedule on the return trip. It is also used in emergency situations, when roads are blocked due to weather or severe accidents for example.

Algorithm:

The user will first set the BusID. The program will try to locate the bus holding this busID. If this busID does not exist, the program will send a message to the user: "Can't do the short turn action because the bus (busID:XXXX) does not exist in this system", and ask the user to enter another busID. If the BusID exists, the program will list to the user all trips which are to be run by this bus. The user can choose one trip from the list. The selected trip is the trip which will be partly cancelled in the short turn action. After selecting the trip, the program will find all control stops of that selected trip, and provide to the user. The user, then, selects one control stop as the short turn point as well as the starting time for the return part trip which will be created by the system. The program will then cancel the part of the selected trip from the short turn point to the destination terminal

as well as the return trip of that bus. It will create a new trip from the short turn point to the destination terminal of the return trip which was cancelled using the trip pattern of this return trip. If the bus was not scheduled for a return trip, the system will look for existing trip pattern of other running buses and ask the user to make a selection before completing the creation of the new return "part trip". The appropriate files will be modified and loaded so that when the simulation will resume, the bus upon its arrival to the specified short turn point, will not continue to run its trip to destination: instead, it will turn back and complete a part trip from that point to its terminal of origin.

The flow chart based on the above algorithm is shown below:

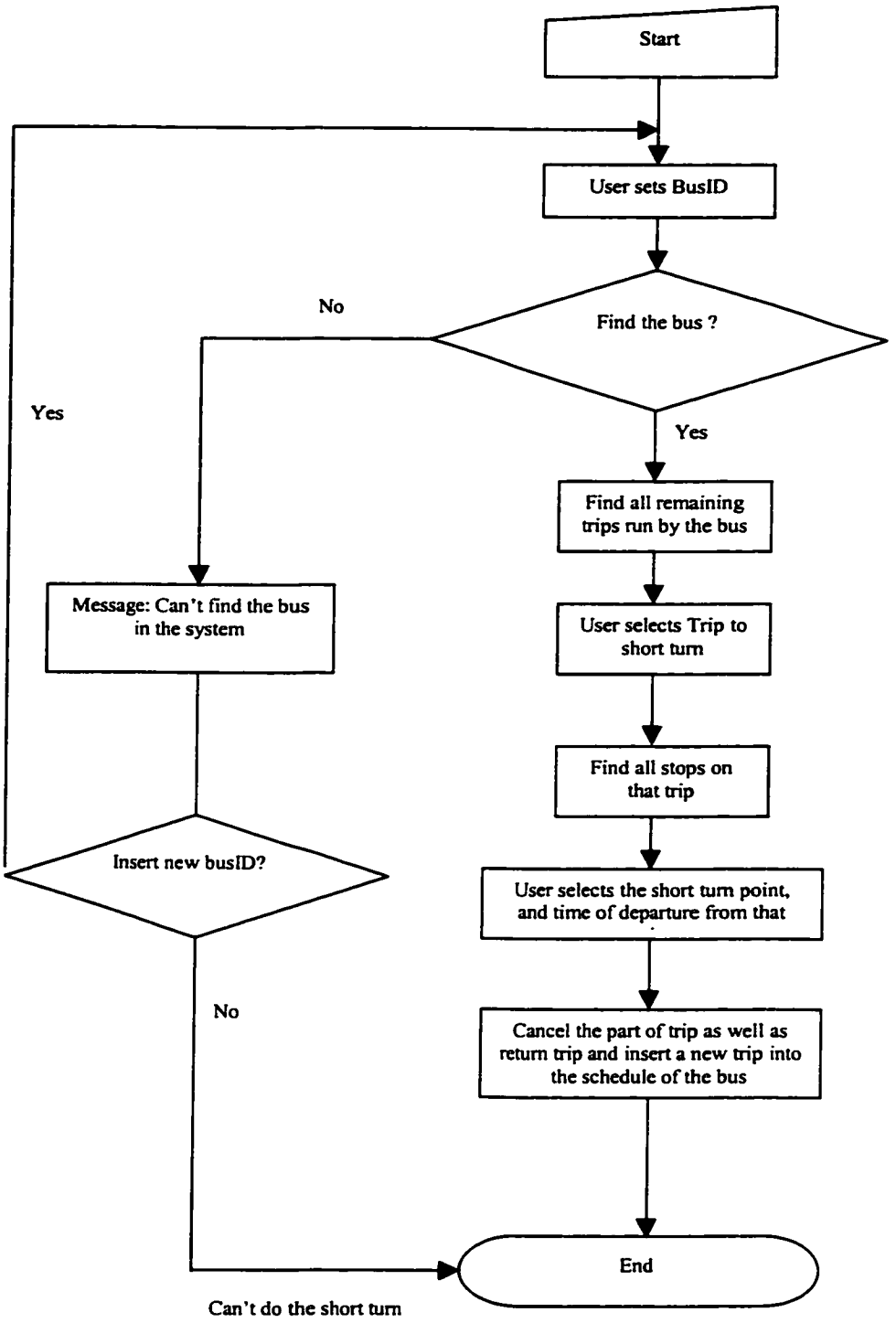


Figure 13: Flow chart for short turn

III-3. A Brief User's Guide for the Central Control Action Module:

The following is a user guide for the Central Control Action module. It is assumed that the user already knows how to use the previous version of the OPSSIM simulation model. Users who are not familiar with OPSSIM should first read the OPSSIM simulation Manual[1].

Requirement:

In order to run the OPSSIM simulation model with the Central Control Action, the user must

- Meet the platform requirements: PC desktop with window95/98/NT/2000 operating system
- Have the enhanced version of OPSSIM simulation model including the central control action module
- Have all data files used to run the OPSSIM simulation model

Installation:

Installation of the new version of OSSIM including the Central Control Action Module does not differ from the installation of the previous version of OPSSIM simulation model. The executable file has the same name as before, OPSSIM.exe, and it must be installed under the same directory as the data files. The new data files that will be generated by the Central Control Action Module will also be created under the same directory.

As mentioned before, while the Main Menu of the Central Control Action module lists four distinct central control actions which have been analyzed earlier in this chapter, only one of these actions, the "Insert Bus Following Breakdown" has been coded and integrated into OPSSIM due to the complexity related to integrating control actions with the existing code of the OPSSIM simulation model.

Start up OPSSIM simulation model:

- Double click the OPSSIM.exe file icon in the Windows Explorer.
- Load scenario.
- Set simulation start and end time, and the other parameters needed by OPSSIM simulation model.
- Run simulation model.

More detailed instruction on how to run the OPSSIM simulation may be found in the OPSSIM simulation manual [1].

We recommend to use the slower simulation clock rate(less than or equal to 20) so that the user has time to see the breakdowns appear on the screen and has the chance to react in a timely way.

Inserting a new bus following a breakdown:

When the simulation starts to run, the run time screen of the simulation is displayed. The run time screen of the OPSSIM simulation model is shown in Figure3.

There is a Radio MSG(Radio Message) field in the run time screen. The OPSSIM simulation model will pop the information regarding bus breakdown into this field. The information includes the busId, error code, the time at which the bus has broken down and the stop where the breakdown took place. Note that a "code 3" represents a total breakdown of the bus which implies it is no longer capable to operate on that day. The other codes relate to various malfunctions but the bus is normally capable to carry out its trips. On the top right of the screen, the simulation clock will show the simulation time. If the simulation is interrupted, the simulation clock will show the time at which the simulation was paused. This information will be used in Central Control Action Module.

When the user finds that one bus has broken down, and wants to insert another bus to replace the broken bus, the following steps must be performed:

- Press F9 function key: This will interrupt the simulation which will be paused. Write down BusId.
- Press F8 function key: It will initiate the Central Control Actions Module. The Main Menu will appear (Figure7).
- Move the cursor to the Insert Bus Following Breakdown item, and press the Return button: The Insert Bus Following Breakdown Menu will appear(Figure9).
- Set the broken bus Id:

- Move the cursor to this item, and press the Return button: the Window for setting the broken bus Id will appear (Figure10).
- Input the bus Id of the broken bus and press the Return key. It will return to the Insert Bus Following Breakdown Menu automatically. The bus id of the broken bus can be obtained from the Radio MSG field in the run time screen.

➢ Run the Insert Bus Following Breakdown program: Move the cursor to the item and press the Return button. As the Insert Bus Following Breakdown program runs, the window for selecting the suggested insertion stops and times will appear (Figure 14).

➢ Choose an insertion stop and time from the list provided by the program by.

- Moving the cursor ("➔") to the insertion option which you want to use, and press the Return button: the mark "X" will be shown on the left, beside the chosen suggestion. The user is free to change this selection at any time before leaving this window. This is done easily by moving the cursor to another insertion option appearing on the screen, and by pressing the Return button: the mark "X" will move to the new selected suggestion, confirming that your choice was reset.

- Press the F10 function key: it will return to the Insert Bus Following Breakdown program. The program will proceed with inserting the new trip based on your selection.

➢ Load the new data files to the simulation or quit the Central Control Action Module:

- Load the new data files: Move the cursor to the "Load Control Action to OPSSIM" item, and press the Return button. The new data files will be loaded to the simulation model, and the simulation will resume. The run time screen of the simulation model will appear again.
- Quit the Central Control Action Module: Move the cursor to the "Cancel Control Action and Return to OPSSIM" item, and press the Return button. The simulation will resume, and the original unchanged data files will be reloaded. The run time screen of the simulation model will appear again.

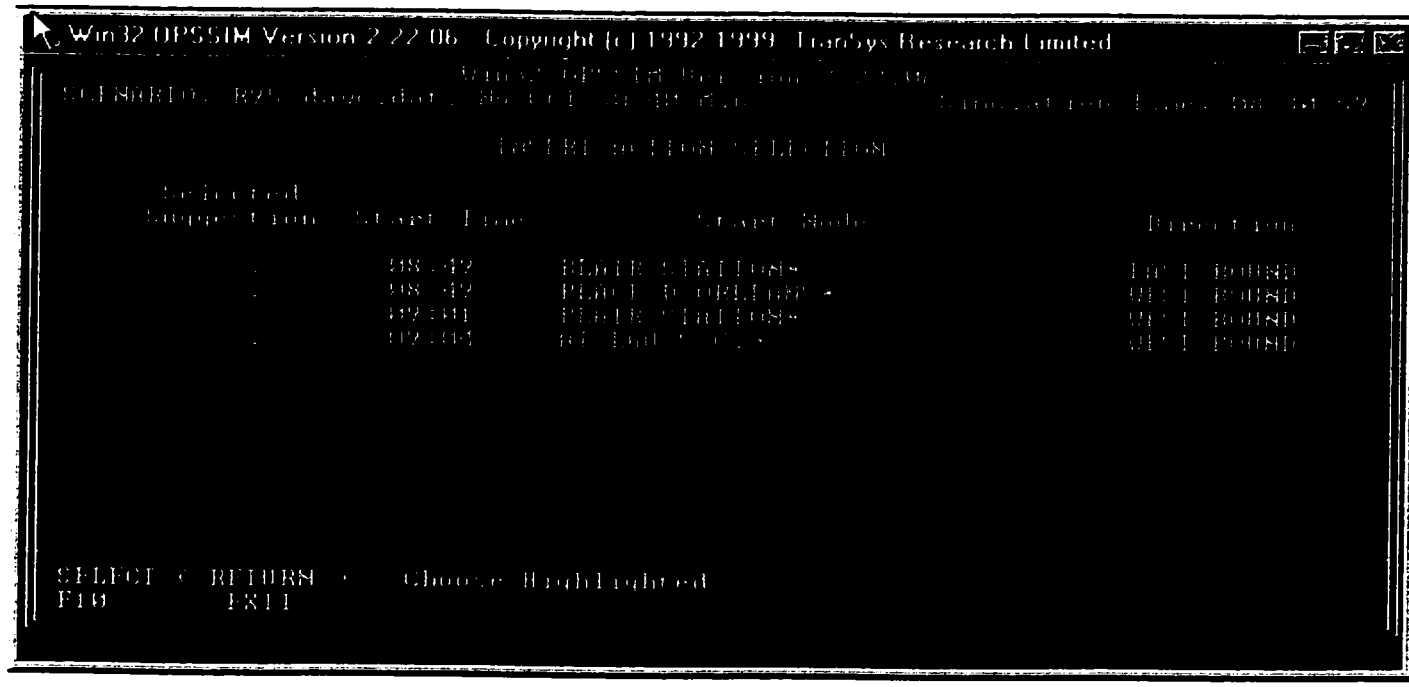


Figure 14: Window for selecting the suggestion

The user guides for the insert trip action, the cancel trip action and short turn a bus action are not included in this thesis: implementing these control actions was beyond the scope of this thesis.

IV. RUN TIME BUS PROGRESS SCREEN:

IV-1. Purpose of the Development of the Run Time Bus Progress Screen:

The purpose of the development of the run time bus progress screen comes from the fact that the run time screen of the OPSSIM simulation model lacks in providing the user key information required by the controller in monitoring the progress of buses along a given route. Without this information, it becomes impossible for the controller to intervene and implement effective control actions to restore service. In this Chapter, we will first introduce the original run time screen, and explain why we need the run time bus progress screen.

The run time screen of the OPSSIM simulation model is the default screen which appears as soon as the simulation starts. It is illustrated on Figure 3. There is more information on the screen. The following will introduce each field on the screen. For convenience, we reproduce Figure 3 below.

Scenario: This shows which scenario is used in the current simulation: there are three possibilities, namely the No Control(R95_dayc.dat, No Ctl), Schedule Control(R95_days.dat, FullDay Sch Ctrl), and Headway Control(R95_dayh.dat, FullDay Head Ctrl).

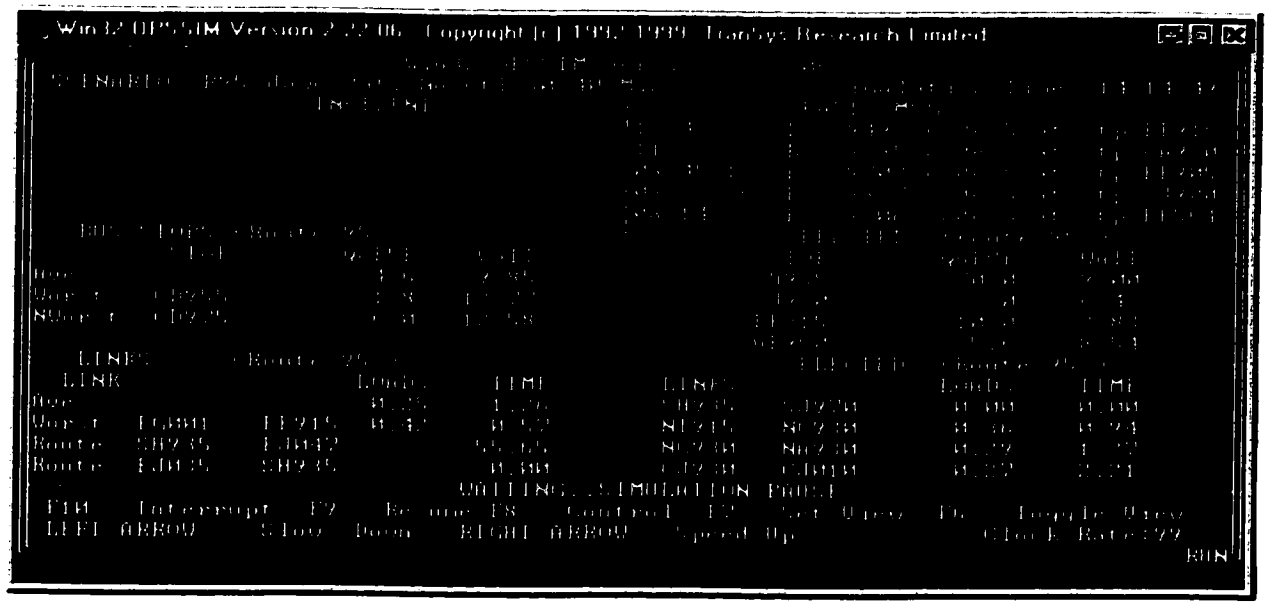


Figure 3 (copy): Run time screen of OPSSIM simulation model

Simulation Time: This indicates the current simulation time. It will be updated every second of simulation time during the simulation run. It is normally not a real time clock, unless the "clock rate" is set at 1.

Clock rate: It is on the right bottom of this screen. It displays the current multiple of real time at which the simulation is running. The range of the clock rate is from 1(real time) to 20(1 minute = 3 seconds), or can be set to 99 that indicates a batch run.

Incidents: This section displays a rolling log of the most recent incidents, such as, bus breakdown. It includes the time of occurrence, end time if over, and a brief description of this incident.

Radio MSG: This area is the broadcast field for reports of equipment failures that normally come to the controller via voice radio in the real world. It indicates breakdown bus Id, the time of breakdown, the type of

breakdown and the location[1]. The type of breakdown is represented by the code number, such as code 3 which means that the bus has a major breakdown and can not operate.

Bus Stop: The stop performance area presents the queue lengths when the bus arrived at the stop, and the passenger wait time averaged over the previous quarter hour interval. The left side presents the statistics of the wait time and queue lengths of the worst, second worst and average of the worst 50 % of stops on the route with the stop id. The right side presents the statistic of the wait time and queue lengths for preselected stops of interest as specified by the user prior to running the simulation[1].

Links: Similarly, the link section presents bus loads (% of capacity) and transit times (minutes) for the worst and average links (left side) and for selected links of interest (right side)[1].

Soft keys: This section is on the bottom of the screen. It shows which function keys are activated during the simulation and which function each key performs. For example, F9 is used to pause the simulation. [1]

While the run time screen provides information on bus incidents, the statistics on bus performance at certain stops and over certain links, this information is insufficient for controllers to allow them to trace buses, anticipate problem situations or make corrective actions. In fact, the use of control actions described in Chapter III will usually be recommended by controllers in response to a specific incident, or to restore effective service when conditions appear to be degrading. In order to monitor these

conditions, more information needs to be made available to the user on-line, particularly information regarding bus progression along the route.

Before we designed the new run time bus progress screen, we went to the OC Transpo Control Centre to investigate the requirements of the run time screen, and at the same time, to view the real run time screen used by central controller in the OC Transpo Control Centre.

According to Mr. G.D. Dean, a superintendent of transportation operations, the most essential information required by central controllers to implement central control on line is:

- The bus performance information at a stop, such as the queue length, and the schedule adherence.
- The bus performance information on the link, such as the loading percentage on the bus, and the link transit time.
- The trace information on each route, such as the location of the bus on a route, and the present headway between buses.

Clearly, the run time screen of the OPSSIM simulation model did not provide users all the necessary information to make useful control decisions.

On the actual run time controller screen in the OC Transpo Control Centre, the three kinds of information listed above are displayed. A very normal graphical interface is used to display information regarding bus location and headway. There is one electronic route map on the controller's desk. All routes are showed using different color lines. Circles on these lines

express nodes in routes. There is one light at each node under the map. The light can be changed in three colors. The Red color indicates that a bus is very late or early. The Green indicates that a bus is on time. The Yellow color indicates that a bus is a little late or early. Each time a bus reaches a node, the light will flash in real time with one of the three colors. At the same time, the bus number will also be shown at the node. This facilitates the job of the controller who monitors the bus performance and traces the buses.

Enlightened by the electronic route map, and given the actual requirement of central controller, we decided to modify the existing run time screen of the OPSSIM simulation to enable it to better monitor bus progress. Discussions with OC Transpo representatives made us realize that it took one year for a team of programmer to design the sophisticated graphic user interface used by the central controllers. Such a project is well beyond the scope of this thesis. Hence we decided to represent the relevant information in a table form in our bus progress screen. Two other reasons supported our choice of the table form: One is that the table form requires significantly less space to display information, and the other is that it is very clear for users to view the information.

IV-2. Design of the Run Time Bus Progress Screen:

IV-2-1. Design Description:

To create the Run Time Bus Progress Screen, we needed to design the screen view which will provide the user with bus progress information during the simulation, as well as the logical algorithms used behind the screen, which locate buses, as the simulation progresses, and display the information to the user through the bus progress screen. In this section, we will introduce the run time bus progress screen and its relationship with the other OPSSIM screens, and then, explain the methodology and algorithms used to support this screen in real time.

IV-2-1-1. The Screen View:

In creating the screen, one must first decide which data will be shown and how to show them. We can't display all the data provided by the AVLC and the APC data files, or generated during the simulation due to space limitation on the bus progress screen. There is also a problem of information overload for the user if too much information is displayed at once. We decided to remain within the existing dimension of the OPSSIM run time screen. Critical information appearing on the run time screen, including the simulation time, the incident and Radio MSE areas, needs to remain on screen at all times, leaving only the lower half of the screen, an area of 11 rows by 80 columns, for the new bus progress screen.

The final selected data will be explained below(Figure 15):

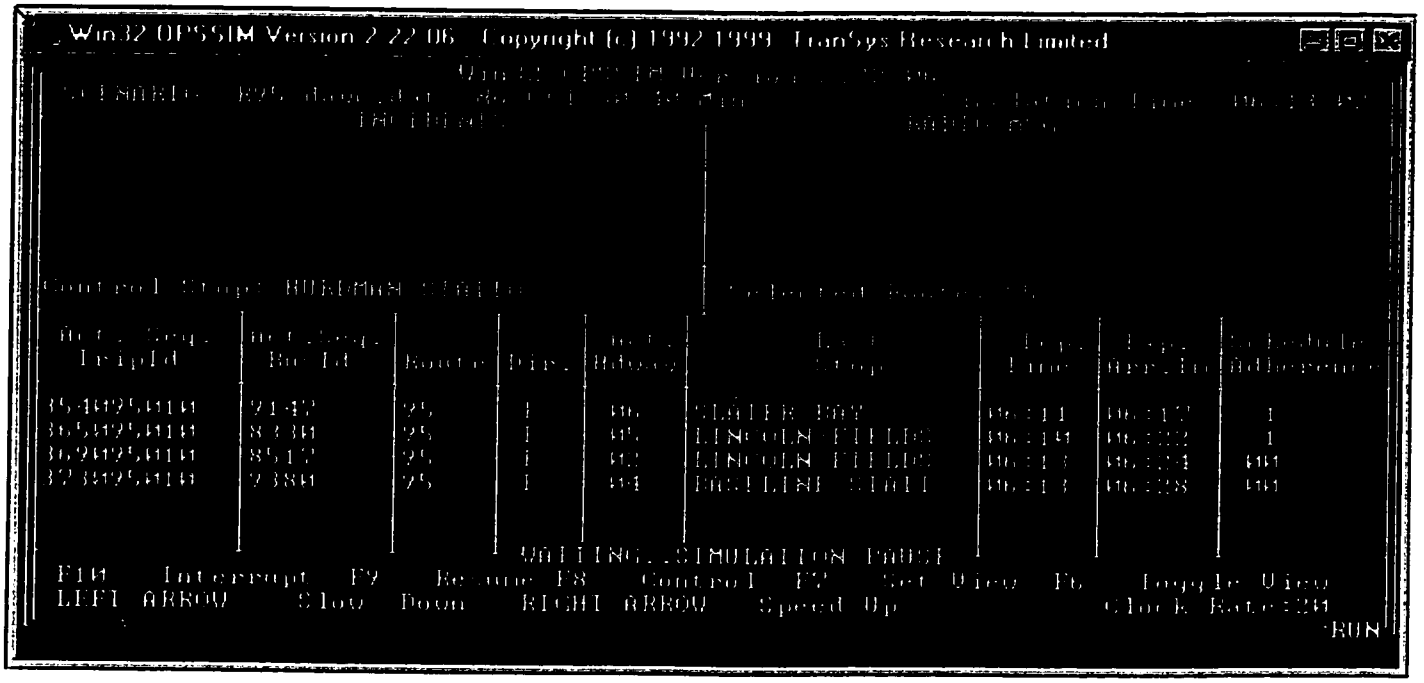


Figure 15: Run Time Bus Progress Screen

From the screen, users can view on-line information about the sequence of buses coming to the control stop. There are six rows in the bus information display table. That means that up to six bus records can be displayed in the table at any time. Buses in the table follow a specific sequence. The bus in the first record line will be the first bus to the control stop. The bus in the last record line is expected to be the last bus among the six buses to the control stop after the present simulation time. The description of each field in the table is below. This screen can be used for headway control, schedule control, and transfer stop control.

Control Stop: The control stop is the observation point from which the performance of a sequence of buses coming to it may be reviewed. The name

of the selected control stop will be shown in this field on the Bus Progress Screen. Users must choose a control stop from the control stop list that is shown in the Choose Control Stop Window[Figure16], before entering the Bus Progress Screen.

Selected Routes: Since the OPSSIM simulation model is currently designed to run a single bus route, there are no route options. The route number shown in this field is the route which is being simulated. By default, it is route 95 because the data files used in the simulation pertain to route 95. Future enhancements of the model may allow for inclusion of more than one route.

Actual Sequence TripId: This field tells the user which trip the bus is running now. The tripId, with its ten digits, can display a lot of information to the user. For example, for the tripId 0486095010, the first four digits express the start time of the trip expressed in minutes of simulation time. 0486 expresses that the start time of the trip is 8: 06 am. The fifth, sixth, and seventh digits from left to right, combined together, express the route number of the trip: 095 expresses that the route number of the trip is 95. The combined eighth and ninth digits from left to right express the trip type code of the trip. (Note: the concept of trip type is defined in Table2 of chapter III): 01 express that this trip is running in the weekday. The last number expresses the direction of the trip. In this example, the last digit is 0. That means the direction of the trip is east bound.

Actual BusId: The data in this field shows which bus is running now.

Route Number: It shows which route the on line bus is running. It will be used later when the OPSSIM simulation supports multiple routes.

Direction: It shows the direction information of the trip. This information can also be obtained indirectly from the tripId for those users who are very familiar with the tripId data format.

Actual headway: The actual headway expresses the time interval between the current bus and its predecessor. It is calculated by the core of the OPSSIM simulation model when the current bus departs from the stop indicated in the "Last Stop" column. These data are very useful for headway control. The controller can know the relative distance between two buses from the data. If the data displayed in this field are in a tolerable range, it means the headway performance of this sequence of buses is good. Otherwise, it indicates bad headway performance and suggests the need for a headway control action in order to improve the spacing between buses.

Last Stop: The last stop provides an indication of where the bus is along the route at the current simulation time. It is the last node the bus has been reported to leave on route for the control stop displayed on the bus progress screen and selected by the user. The name of this last stop is shown in this field.

Departure time from the last stop: The data in this field shows the actual departure time of the bus from the stop shown in the "last stop" column. These data are also provided for the controller to trace the bus along the route.

Expected Arrival Time to the Control Stop: The data in this field shows the expected arrival time for each bus to the control stop. The expected arrival time of the first bus in the table is the scheduled arrival time of that bus to the control stop. For the other five buses, their expected arrival time is calculated by taking the expected arrival time to the control stop of the previous bus plus the headway between the current bus and the previous bus. Note that times displayed on the table are rounded to the nearest minute.

Schedule Adherence: The schedule Adherence column shows the difference between the actual departure time from the stop in the "last stop" column and the scheduled departure time from that stop for that bus. It is also calculated by the core of the OPSSIM simulation model. A negative value indicates that the bus is running faster than the schedule, while a positive value represents the lateness of the bus. This data is provided to support schedule control. The controllers can know whether the bus is on time, early, or late.

IV-2-1-2. The input for the new run time bus progress screen:

IV-2-1-2-1. Input data files:

R95_stop.dat file is the only input data file used in the run time bus progress screen program. The schema of this data file is shown in the following table:

Field Name	Data Type	Content	Example	Notes
nodeId	integer	The identifier of the node.	7010	
stopCode	character	The code that represents each stop.	SH935	SH935: the eastbound platform of Baseline station
serialNum	integer	The sequence number of stops in the direction.	1	1: the first stop in the direction
nDir	integer	Displays which direction the bus is heading.	0	0: east bound 1: west bound
StopName	character	The name of the stop	BASELINE/STATION	

Table 6: the schema of the R95_stop.dat file

The nodeId is the only required external parameter for the run bus progress screen. This parameter provides the nodeId of the node the user chooses to monitor during the simulation, and where the bus progress information will be generated.

IV-3. Navigating Between the Bus Progress Screen and the Run-Time Screen:

There are two ways to switch from the run time screen to the Bus Progress Screen.

One way, called **toggle view**, is to press the F6 key. Pressing the F6 key, puts the simulation on hold while the Control Stop Selection

Window(Figure 16) appears. After the user selects the control stop and presses the F10 key, the run time screen switches to the bus progress screen and the simulation resumes.

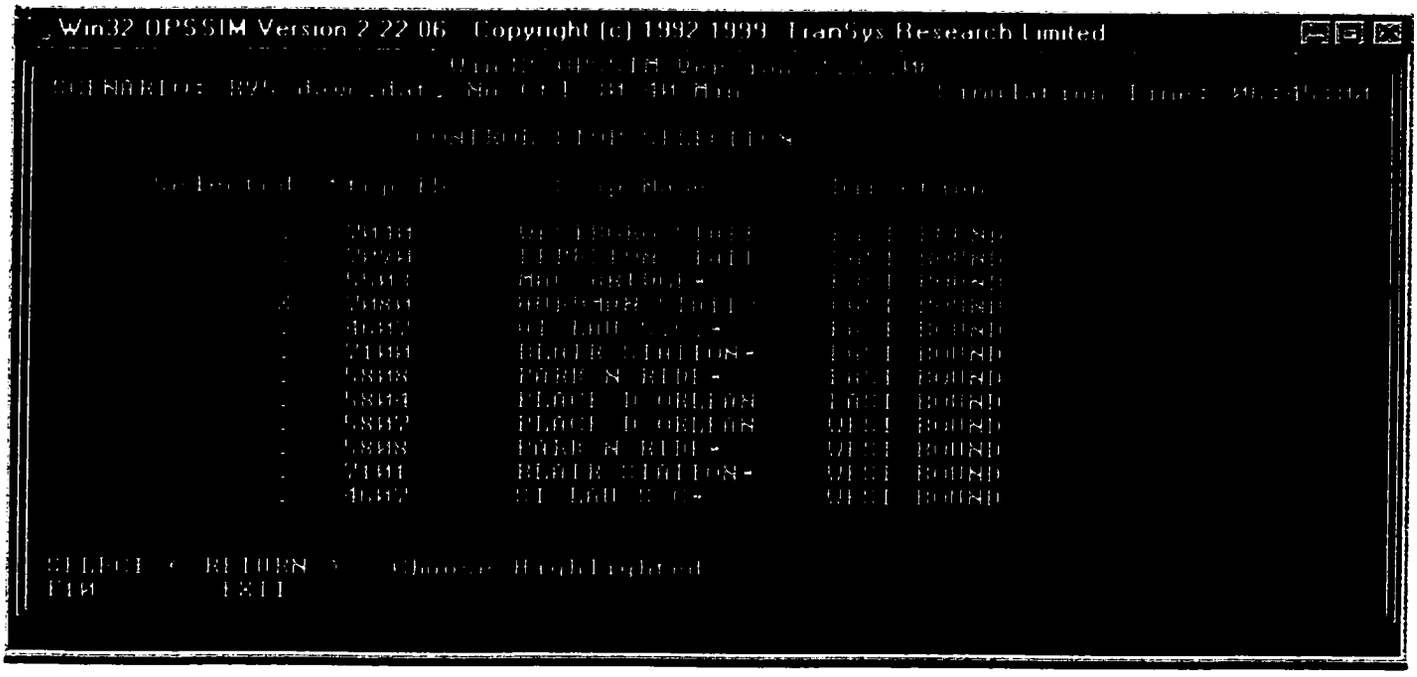


Figure 16: Control Stop Selection Window

Control Stop Selection Window: There are four columns in the window. The first displays whether the control stop has been selected("X" mark) or not("." mark), the second indicates the stop Id, the third shows the stop name, and the last provides the direction of the stop. There is only one

cursor in the window. Users can use the UpArrow or DownArrow keys to move the cursor up or down.

Scrolling down the window: If the cursor is on the last line displayed on the screen, and the user keeps pressing the DownArrow key, the window will scroll down one line, and the name of the following stop in the list will be shown. If the window does not change, it means the stop in the last displayed line is the bus terminal, and there are no more stops after it.

Scrolling up the window: If the cursor is on the top line displayed on the screen, and the UpArrow key is pressed, the window will scroll up. The name of the stop preceding the stop currently being displayed will appear unless the stop displayed on the top line is the bus terminal, in which case the window doesn't scroll.

Selecting the control stop: To select a stop as control stop for the bus progress screen, move the cursor to the line indicating the name of the stop chosen to be the control stop using the arrow keys, and press the Return button. The "selected" mark (X), will appear in the first column of that row.

Changing the selected stop: If the user has set the control stop and wishes to change it to another stop before leaving the Control Stop Selection Window, the user can move the cursor to the line displaying the name of the stop that will be the new control stop, and press the Return button. The "selected" mark [X] in the first column will move to the line indicating the name of the new selected control stop.

Returning to the Run Time Bus Progress Screen: After setting the control stop, press the F10 key: the Bus Progress Screen will appear, and the simulation resumes running immediately.

Back to the run time screen: Under the bus progress screen, by pressing the F6 key, the screen will be returned immediately to the run time screen.

Another way to select the Bus Progress Screen, called **Set View**, is to press the F7 key. Pressing the F7 key, puts the simulation on hold while the Set View Window(Figure 17) appears.

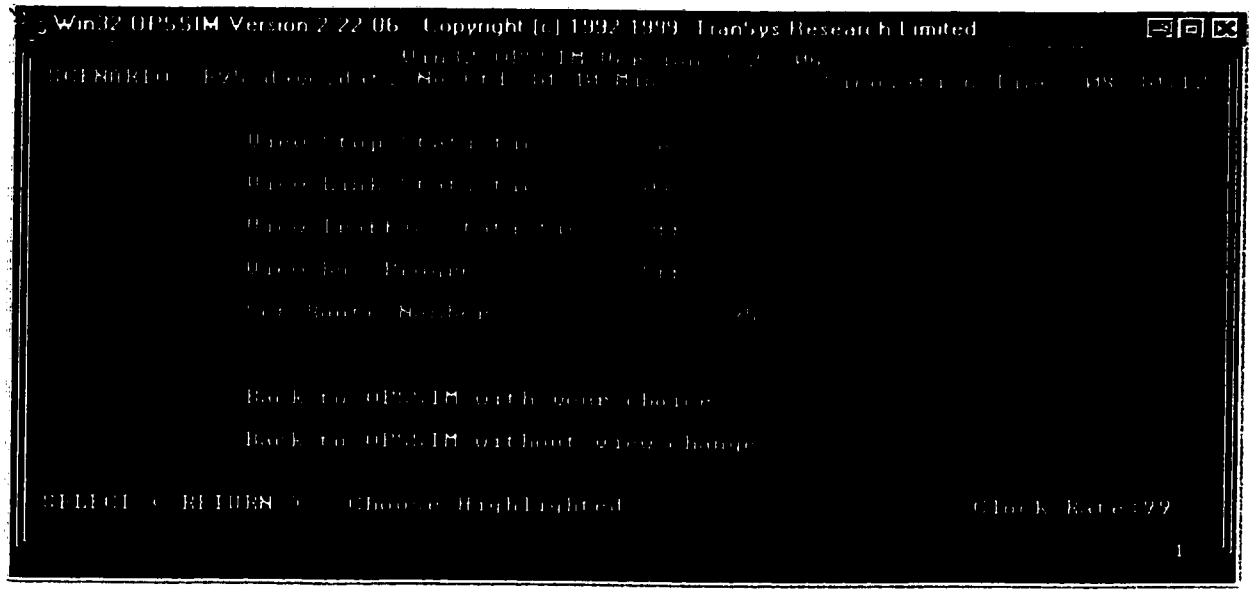


Figure 17: Set View Window

Set View Window: There are seven information lines in the window. They are :

- View Stop Statistics
- View Link Statistics

- View Traffic Statistics (Not relevant to this thesis, but used by Transys)
- View Bus Progress
- Set Route Number (Not relevant to this thesis, but used by Transys)
- Back to OPSSIM with your choice
- Back to OPSSIM without view change

There is also one cursor in the window. Users can move the cursor to any information line using the UpArrow or DownArrow keys.

Selecting View: (Stop Statistics, Link Statistics, and Bus Progress): By moving the curser up or down the selection list, the user can select the views for the simulation screen. By pressing the Return button, the indicator will change from "On" to "Off" and vice versa. To view Stop and/or Link Statistics, the indicator for "View Bus Progress" must be "Off". If the "View Bus Progress" indicator is "On", the Bus Progress Screen will appear once the simulation is resumed. To return to the simulation, bring the cursor either to "Back to OPSSIM with your choice" to return to the screen which has been selected, or "Back to OPSSIM without view change" to return to view screen in place before the Set View Window was invoked.

In here, this program doesn't allow users to view link or stop statistics and bus progress information on the same screen due to the screen size limitation.

There are more windows linked to the Run Time Bus Progress Screen Window. In order to avoid any confusion, we draw the following map. Hopefully it can provide assistance to users navigating between various windows.

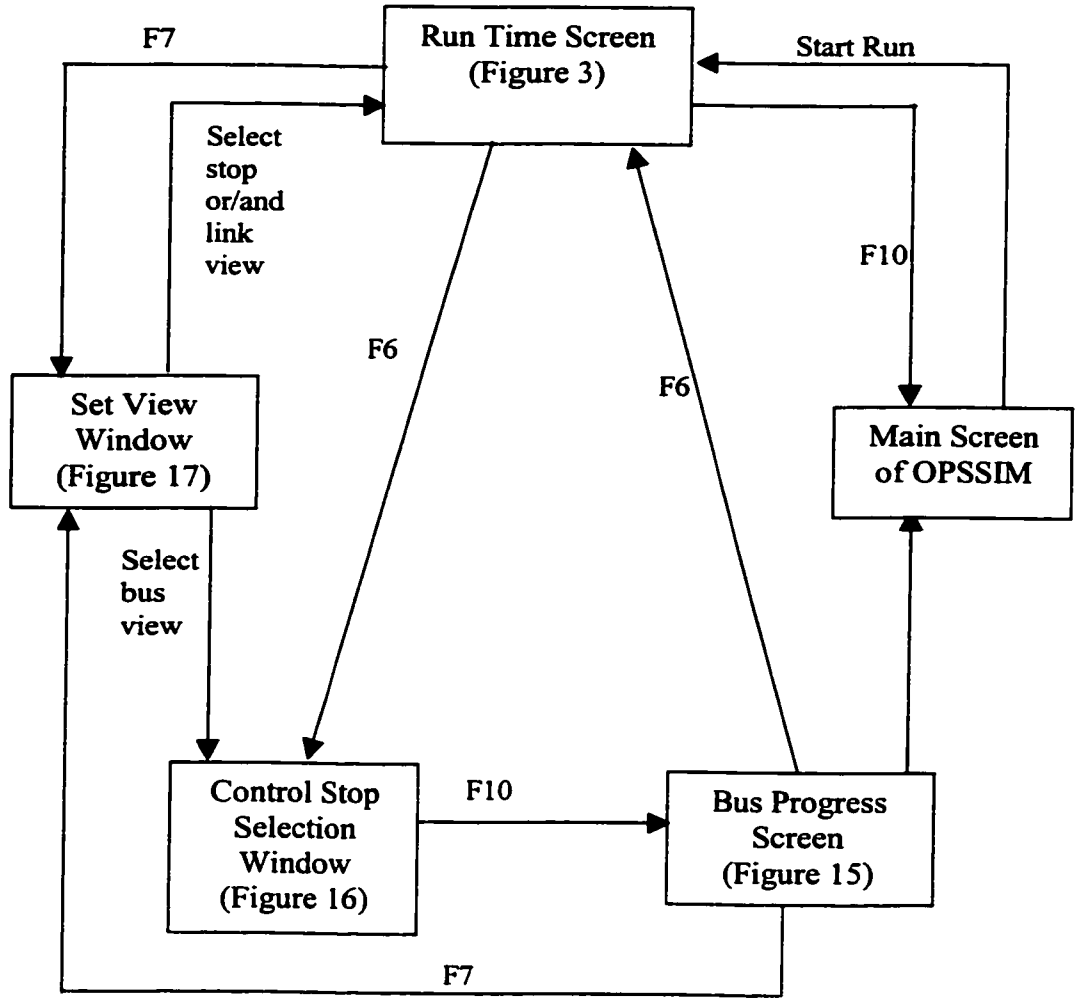


Figure 18: Map of five windows

IV-4. The Algorithm:

Finding Buses Coming to the Control Stop:

In order to make the run-time bus progress screen work, the user will first select the control stop, as well as the direction being followed by the buses to be monitored (i.e. Eastbound or Westbound). The program then starts to identify the sequence of buses heading for the control stop.

For ease of exposition, let us assume that the nodes along the route are numbered sequentially from 1 to n, where node n is the selected control stop while node 1 is the start terminal of the trips run by buses heading towards the control stop in the direction specified by the user. The algorithm must construct a sequence of buses S to display on the bus progress screen. Let us define r as the remaining number of buses to include in S, with r=6 at the beginning. The algorithm starts with node n-1, and works its way back to n-2, n-3 and so on, until six buses have been identified for inclusion on the bus progress screen. Assume that the algorithm is now at some node i, where $1< i \leq n-1$. The algorithm constructs a list L_i of BusID of all buses who have left stop i, but who have not yet left stop i+1. We say List L_i contains a_i entries. The list is sorted by increasing departure time from node i. If $a_i \geq r$, the first r buses on this list will be placed in that order at the tail of sequence S, and S is complete. Otherwise, the a_i entries in L_i are placed at the tail of sequence S, we set $r= r-a_i$ and $i = i-1$, and we repeat this procedure until r =0.

Refresh the Run Time Bus Progress Screen Table:

The information in the Run Time Bus Progress Screen Table should always be updated as the simulation generates new information. Thus, the user can always rely on the latest data quickly.

When any bus arrives at any stop, the performance statistics data and the bus location data will be updated in the OPSSIM simulation. Hence, the data in the bus progress table will be refreshed each time any bus arrives at any node during the simulation.

The flow chart based on the above algorithm is shown below:

Note:

BPSi: denotes the various test cases performed in Chapter V(section V-3-1). Here, BPSi represents the ith test case on the run time Bus Progress Screen test plan. The detailed description for this test plan is presented in Chapter V.

Finally, a description of the main functions in the Bus Progress Screen program are found in Appendix B, while data structures are provided in Appendix C in order to assist future program developers in their efforts to introduce new features and enhancements to the simulation model.

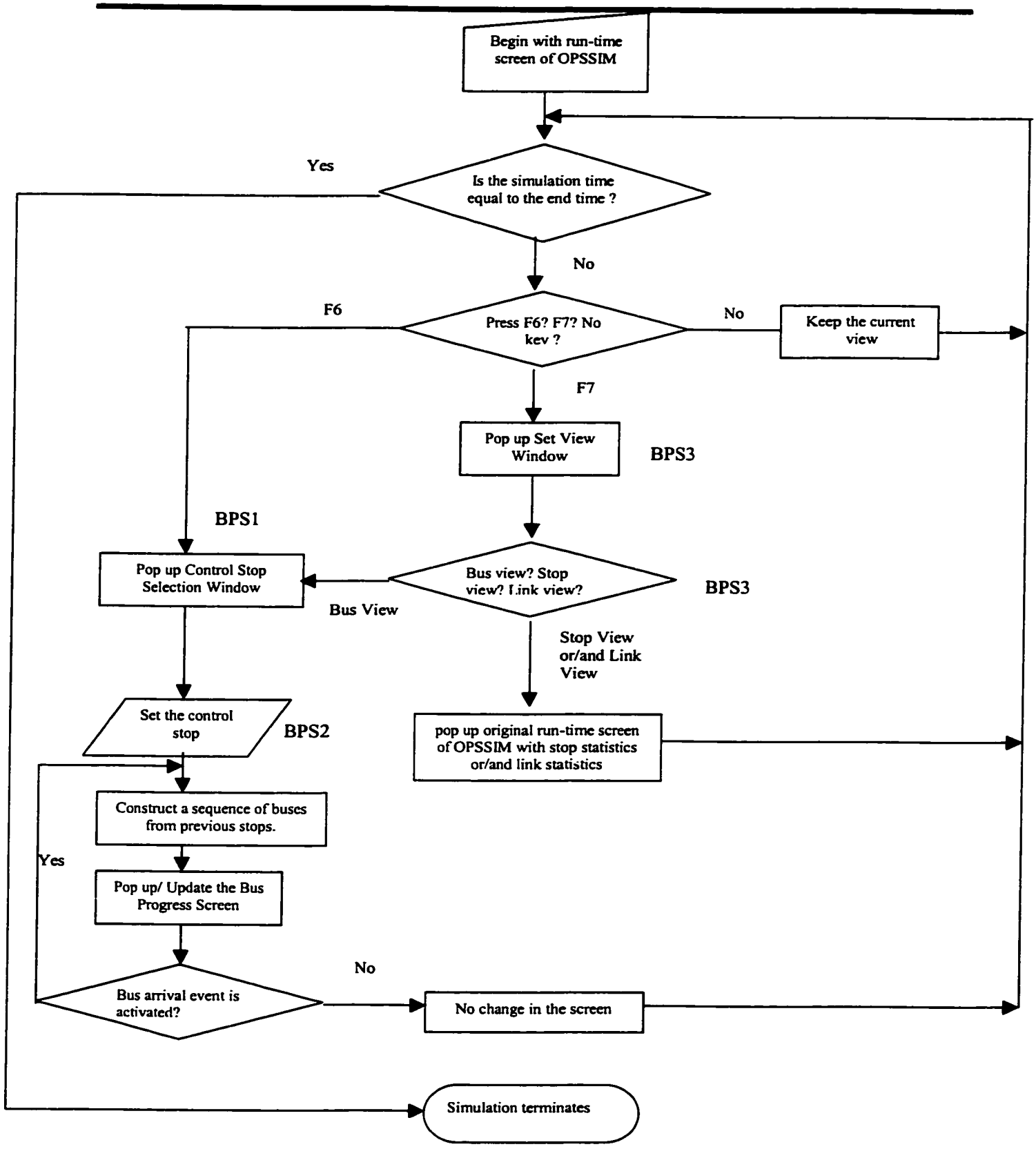


Figure 19: Flow chart of the algorithm of Bus Progress Screen

V. VALIDATION TESTS AND RESULTS:

V-1. Scope of the Validation:

A number of tests have been designed for the purpose of debugging the code of the new procedures developed within this thesis, and to validate the results obtained by the new modules. Tests were performed to validate the following new elements of the OPSSIM simulation model:

- The Main Menu of Central Control Action Module (CCAM).
- The user interface associated with the Insert Bus Following Breakdown procedure.
- The correctness of the Insert Bus Following Breakdown procedure.
- The user interface associated with the Bus Progress Screen (BPS).

We use the notation CCAMi and BPSi to denote the i^{th} test done on CCAM and BPS respectively. See also the flow charts on Figure8 and Figure 19.

V-2. Tests on the Central Control Actions Module:

The following tests have been conducted in order to evaluate the user-friendliness of the new screens, and to validate the data generated by the new procedures.

V-2-1 User-friendliness of CCAM:

CCAM1: Tests to make sure that the user interface window of the CCAM will pop up with the correct menu items when requested by the user.

CCAM2: Tests to make sure that the user interface window of the Insert Bus Following Breakdown program will pop up with the correct menu items when requested by the user.

CCAM3: Tests to make sure that the interface allowing the user to enter the BusID of the bus which broke down works correctly.

For all these tests, correct as well as erroneous inputs have been entered on purpose to see how the program would react, and to fine tune the various error messages which appear to the user indicating what was the probable cause of the error and allowing the data to be re-submitted. All these tests have been passed successfully.

V-2-2 Validation of the Algorithms:

The most critical validation of the CCAM, denoted by CCAM4, was performed to make sure that the Insert Bus Following Breakdown procedure works correctly. Tests were done to insure that the bus insertion was valid (i.e. extra bus available, time and location of insertion validated so that the control action is feasible and can be implemented). The test also verifies that the program returns correctly to the simulation once a valid bus insertion is

confirmed by the user. We verified that the new files, trip1.dat, block1.dat, trippat1.dat and trippatvc1.dat were created with the correct data. The new code passed all these tests successfully.

We next decided to conduct an experiment to verify that the simulation model read the new files correctly, and that the inserted bus was indeed included in the simulation, and was having some form of measurable impact on the various performance measures as reported on the simulation output file. These experiments are described below.

V-2-2-1. Validation from the output file of the OPSSIM simulation model:

The following series of small experiments will show how the overall bus performance will be affected by the insertion of a new bus following a bus breakdown.

We randomly generated 30 samples through changing the seed number. This seed number is used by the OPSSIM simulation model to generate the random data needed by the system. Of those 30 samples, there were 10 samples generated under the Headway Control scenario, 10 samples generated under the Schedule Control scenario, and 10 samples generated under the No Control scenario.

Statistics on average Headway, On Time Percentage, and Trip Travel Time have been computed for the period between 9:00 am and the simulation end time. This comes from the simulation output file which separates statistics by time of day. Between 9:00 and 15:30, the planned

headway for Route 95 is 5 minutes over most links, except links between Blair and Orleans where the headway is 10 minutes.

When we do the experiment, we put the result data into two sets. The first set in the "No Insertion" file, which holds statistics on the average trip time, headway as well as the average percentage of buses on time over all stops over a pre-defined period of time. These data are generated by the OPSSIM simulation model for the case where no new bus was inserted following a bus breakdown. The other set is the "Insert a Bus" set, which holds statistics but for the case where a new bus was inserted following the bus breakdown. Comparing these two sets of data allows us to evaluate the impact of the central control action.

We put all experiment data into the following tables, and also generated a number of charts based on these data, which allow the reader to better appreciate the experiment results.

Table 7: the experiment result(Average Trip Time):

Seed Number	Broken Time	Control Strategy	Simulation Start and End Time	Without Inserted Bus	With Inserted Bus	Difference
29999999	10:13	N	7:00-14:00	53.71	53.23	-0.48
34567802	12:43	N	8:00-15:00	54.84	54.36	-0.48
19992305	9:10	N	7:00-14:00	55.60	54.34	-1.26
54691233	12:43	N	7:00-14:00	55.79	55.71	-0.08
19926375	9:17	N	8:00-14:00	53.55	52.81	-0.74
12257383	9:42	N	8:00-15:00	54.11	53.74	-0.37
18237525	13.07	N	8:00-15:00	53.98	53.89	-0.09
54326789	8:55	N	8:00-15:00	53.81	53.74	-0.07
13748721	10:10	N	8:00-15:00	53.79	53.51	-0.28
35209874	9:53	N	8:00-15:00	53.85	53.54	-0.31
73789802	8:12	H	8:00-15:00	54.37	53.41	-0.96
38567891	8:19	H	8:00- 15:00	54.25	53.91	-0.34
65588867	11:33	H	8:00- 15:00	54.62	54.53	-0.09
84321192	11:53	H	8:00- 15:00	54.53	53.23	-1.3
76430529	9:12	H	8:00- 15:00	53.79	53.13	-0.66
22670730	8:33	H	8:00- 15:00	54.57	53.39	-1.18
07674228	9:09	H	8:00- 15:00	54.87	53.95	-0.92
88888888	8:59	H	8:00- 15:00	55.15	53.58	-1.57
23456789	8:24	H	8:00- 15:00	55.42	54.43	-0.99
33333333	9:41	H	8:00- 15:00	55.28	54.08	-1.2
00462745	9:04	S	8:00- 15:00	54.75	54.61	-0.14
36952376	8:34	S	8:00- 15:00	54.86	54.40	-0.46
25987523	8:23	S	8:00- 15:00	55.09	54.89	-0.2
63179454	9:03	S	8:00- 15:00	54.40	53.47	-0.93
40111267	9:54	S	8:00- 15:00	54.27	53.38	-0.89

64896302	8:25	S	8:00- 15:00	54.28	54.26	-0.02
56788765	9:04	S	8:00- 15:00	55.62	54.75	-0.87
79841234	8:51	S	8:00- 15:00	55.33	54.50	-0.83
67630207	8:18	S	8:00- 15:00	55.97	54.82	-1.15
53940865	11:56	S	8:00- 15:00	54.80	54.61	-0.64

Note:

N: No Control Strategy

H: Headway Control Strategy

S: Schedule Control Strategy

Difference = Average Trip Time(with Inserted Bus) – Average Trip Time (without Inserted Bus)

Table 8: The Experiment Result(Average Headway):

Seed Number	Broken Time	Control Strategy	Simulation Start and End Time	Without Inserted Bus		With Inserted Bus		Difference	
				Baseline -Blair	Blair-Orleans	Baseline -Blair	Blair-Orleans	Baseline -Blair	Blair-Orleans
29999999	10:13	N	7:00-14:00	5.15	10.41	4.95	9.96	-0.2	-0.45
34567802	12:43	N	8:00-15:00	5.14	10.20	5.04	9.76	-0.1	-0.44
19992305	9:10	N	7:00-14:00	5.63	10.42	5.35	9.42	-0.28	-1
54691233	12:43	N	7:00-14:00	5.26	10.63	5.23	10.62	-0.03	-0.01
19926375	9:17	N	8:00-14:00	5.63	11.71	4.96	9.96	-0.67	-1.75
12257383	9:42	N	8:00-15:00	5.33	10.43	5.02	10.29	-0.31	-0.14
18237525	13.07	N	8:00-15:00	5.75	10.51	5.01	10.21	-0.74	-0.3
54326789	8:55	N	8:00-15:00	5.45	10.39	5.10	10.05	-0.35	-0.34
13748721	10:10	N	8:00-15:00	5.55	10.72	4.97	10.36	-0.58	-0.36
35209874	9:53	N	8:00-15:00	5.59	10.56	5.07	9.87	-0.52	-0.69

73789802	8:12	H	8:00-15:00	5.69	10.53	5.38	10.07	-0.31	-0.46
38567891	8:19	H	8:00-15:00	5.45	10.16	5.06	9.95	-0.39	-0.21
65588867	11:33	H	8:00-15:00	5.54	11.55	5.39	10.64	-0.15	-0.91
84321192	11:53	H	8:00-15:00	5.38	11.25	5.13	10.35	-0.25	-0.9
76430529	9:12	H	8:00-15:00	5.46	10.97	5.05	9.91	-0.41	-1.08
22670730	8:33	H	8:00-15:00	5.67	11.27	5.23	10.29	-0.44	-0.98
07674228	9:09	H	8:00-15:00	5.61	11.05	5.30	10.16	-0.31	-0.89
88888888	8:59	H	8:00-15:00	5.71	11.12	5.34	10.09	-0.37	-0.03
23456789	8:24	H	8:00-15:00	5.66	10.87	5.21	10.11	-0.45	-0.76
33333333	9:41	H	8:00-15:00	5.59	10.71	5.08	10.24	-0.51	-0.67
00462745	9:04	S	8:00-15:00	5.14	9.67	5.02	9.49	-0.12	-0.18
36952376	8:34	S	8:00-15:00	5.86	10.73	5.10	10.12	-0.76	-0.61
25987523	8:23	S	8:00-15:00	5.21	9.84	5.07	9.84	-0.14	0
63179454	9:03	S	8:00-15:00	5.14	10.30	5.08	10.28	-0.06	-0.02
40111267	9:54	S	8:00-15:00	5.34	9.98	5.13	9.95	-0.21	-0.03
64896302	8:25	S	8:00-15:00	5.29	10.02	5.20	9.90	-0.09	-0.12
56788765	9:04	S	8:00-15:00	5.66	10.84	5.21	10.01	-0.45	-0.83
79841234	8:51	S	8:00-15:00	5.54	10.78	5.17	9.93	-0.37	-0.85
67630207	8:18	S	8:00-15:00	5.69	10.71	5.24	10.06	-0.45	-0.65
53940865	11:56	S	8:00-15:00	5.56	10.79	5.27	10.35	-0.29	-0.44

Note:

N: No Control Strategy

H: Headway Control Strategy

S: Schedule Control Strategy

Difference = Average Headway (with Inserted Bus) – Average Headway (without Inserted Bus)

Table 9: The Experiment Result(Average Percentage of Buses on Time):

Seed Number	Broken Time	Control Strategy	Simulation Start and End Time	Without Inserted Bus	With Inserted Bus	Difference
29999999	10:13	N	7:00-14:00	5.40	7.51	2.11
34567802	12:43	N	8:00-15:00	13.90	13.92	0.02
19992305	9:10	N	7:00-14:00	10.44	13.76	3.32
54691233	12:43	N	7:00-14:00	5.44	5.60	0.16
19926375	9:17	N	8:00-14:00	17.48	17.52	0.04
12257383	9:42	N	8:00-15:00	15.53	16.03	0.5
18237525	13.07	N	8:00-15:00	10.93	11.69	0.76
54326789	8:55	N	8:00-15:00	9.99	11.02	0.03
13748721	10:10	N	8:00-15:00	11.87	12.57	0.7
35209874	9:53	N	8:00-15:00	12.01	13.51	1.5
73789802	8:12	H	8:00-15:00	14.62	15.48	0.84
38567891	8:19	H	8:00- 15:00	13.86	15.67	1.81
65588867	11:33	H	8:00- 15:00	16.42	18.52	2.1
84321192	11:53	H	8:00- 15:00	14.22	14.28	0.06
76430529	9:12	H	8:00- 15:00	17.72	20.70	2.98
22670730	8:33	H	8:00- 15:00	15.89	17.45	1.56
07674228	9:09	H	8:00- 15:00	14.72	16.88	2.16
88888888	8:59	H	8:00- 15:00	13.57	18.03	4.46
23456789	8:24	H	8:00- 15:00	14.01	16.23	2.22
33333333	9:41	H	8:00- 15:00	15.31	17.09	1.78
00462745	9:04	S	8:00- 15:00	11.82	14.06	2.24
36952376	8:34	S	8:00- 15:00	17.76	22.36	4.6
25987523	8:23	S	8:00- 15:00	17.10	17.47	0.37

63179454	9:03	S	8:00- 15:00	16.73	17.92	1.19
40111267	9:54	S	8:00- 15:00	16.69	18.89	2.2
64896302	8:25	S	8:00- 15:00	16.89	18.70	1.81
56788765	9:04	S	8:00- 15:00	16.02	19.01	2.99
79841234	8:51	S	8:00- 15:00	15.99	18.09	2.1
67630207	8:18	S	8:00- 15:00	15.67	17.85	2.18
53940865	11:56	S	8:00- 15:00	16.94	18.16	1.22

Note:

N: No Control Strategy

H: Headway Control Strategy

S: Schedule Control Strategy

Difference = Average Percentage of Buses on time (with Inserted Bus) – Average Percentage of Buses on time (without Inserted Bus)

In order to more clearly display the change of the data in with and without central control action systems, we draw the following charts based on the data in Tables 7, 8 and 9.

Charts in Figure 20 describe the change of actual trip time data with and without inserting a bus into the system for the different scenarios. Figure20-a depicts the No Control scenario, Figure20-b the Headway Control scenario, and Figure20-c the Schedule Control scenario.

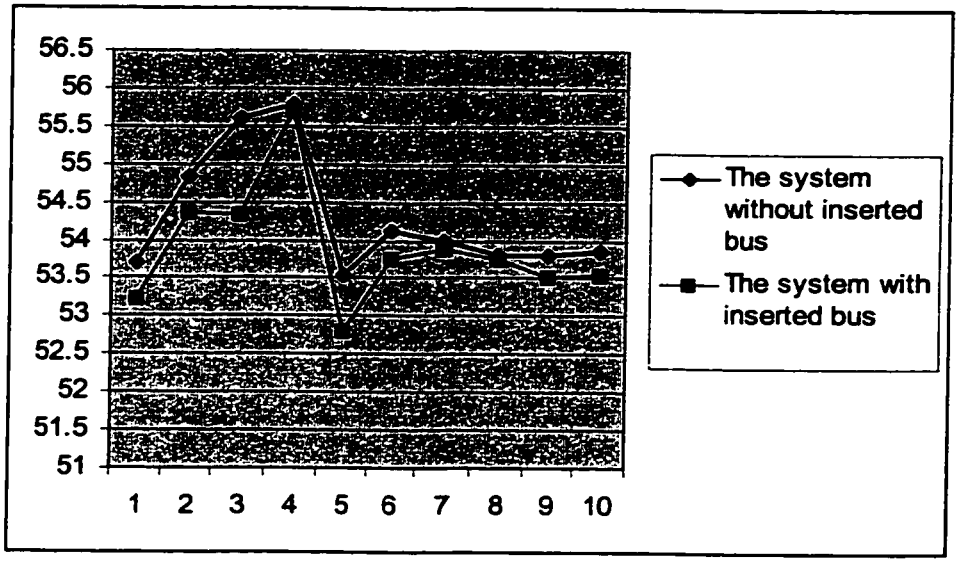


Figure 20-a: Actual average trip time in the No Control scenario

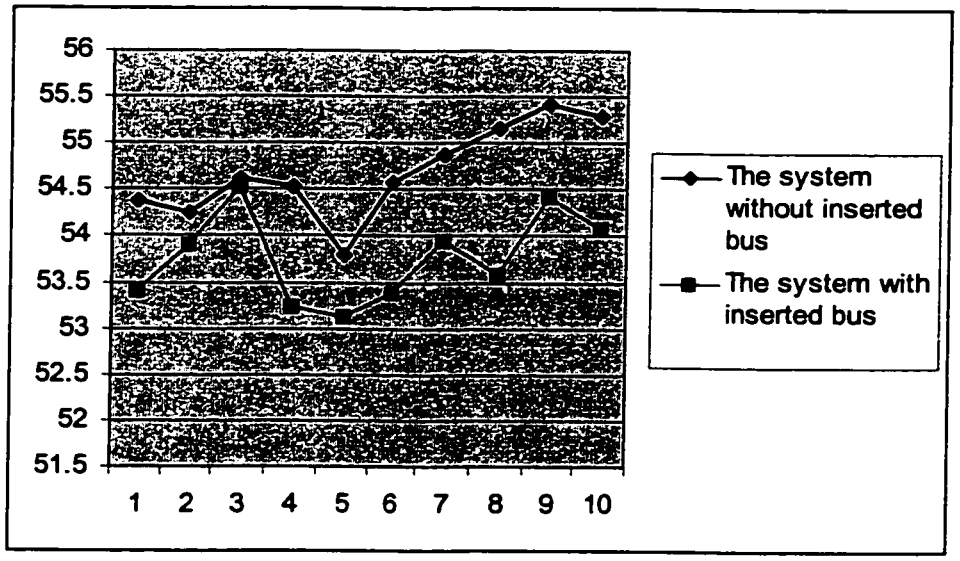


Figure 20-b: Actual average trip time in the Headway Control scenario

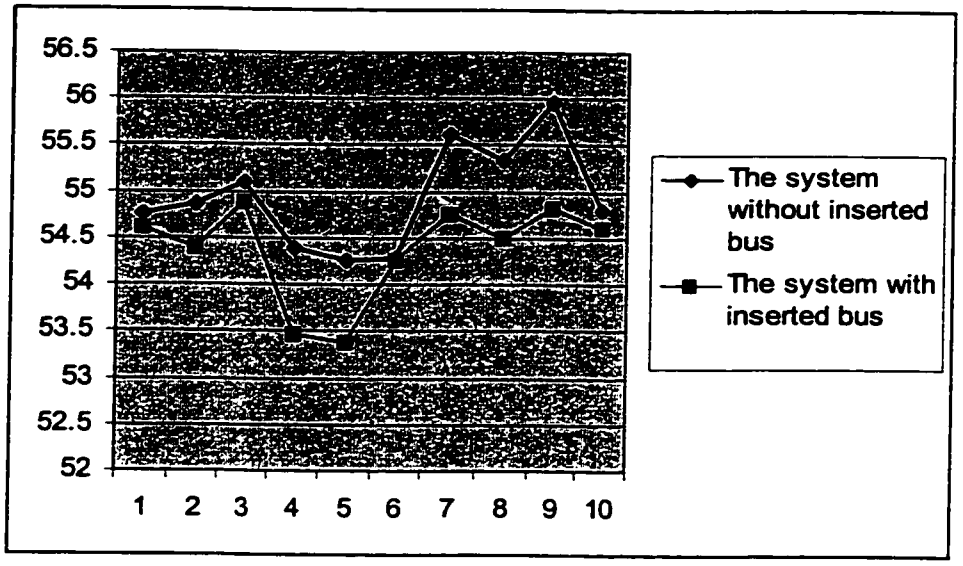


Figure 20-c: Actual average trip time in the Schedule Control scenario

Charts in Figure 21 describe the change in actual average headway with and without inserting a bus into the system for the different scenarios. Figure21-a depicts the No Control scenario, Figure21-b the Headway Control scenario, and Figure21-c the Schedule Control scenario.

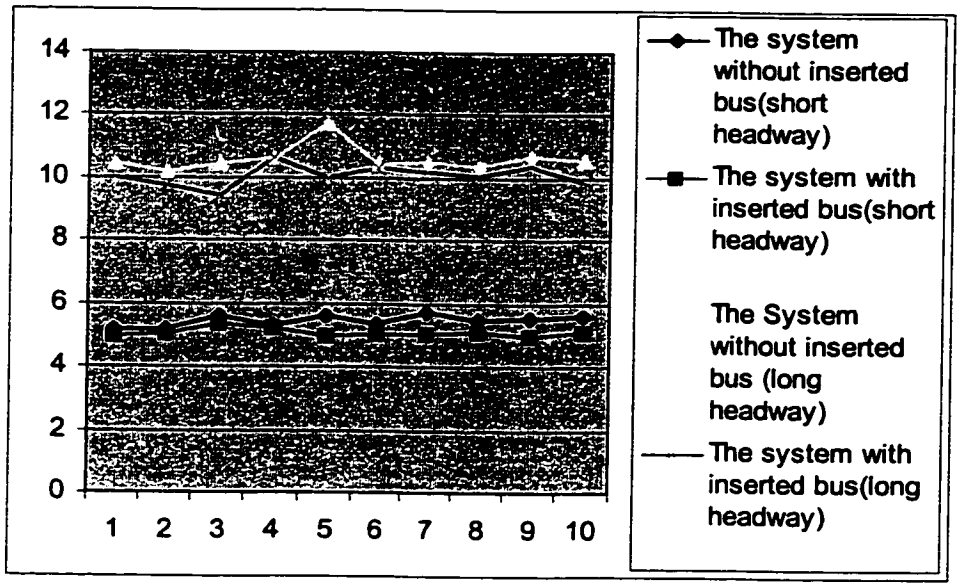


Figure 21-a: Actual average headway in the No Control scenario

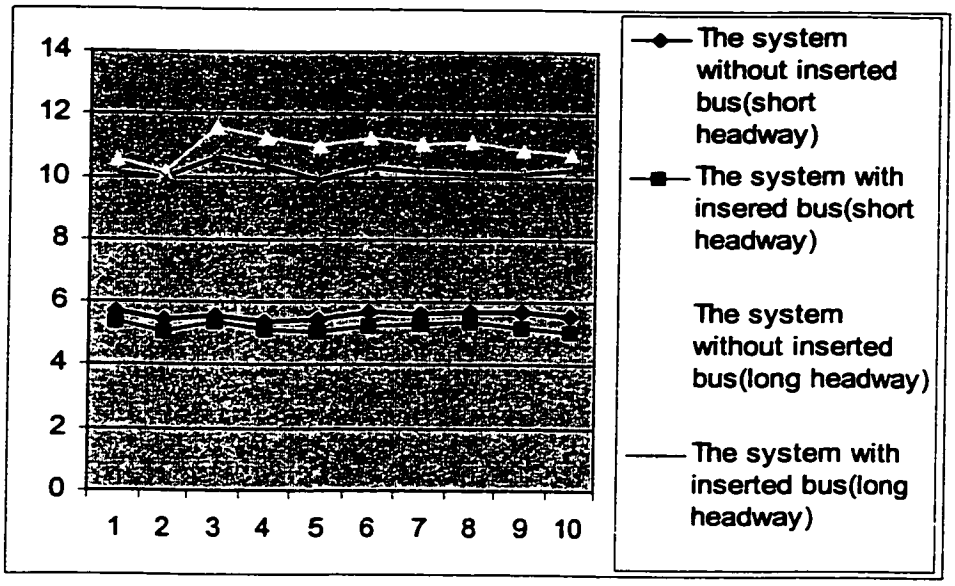


Figure 21-b: Actual average headway in the Headway Control scenario

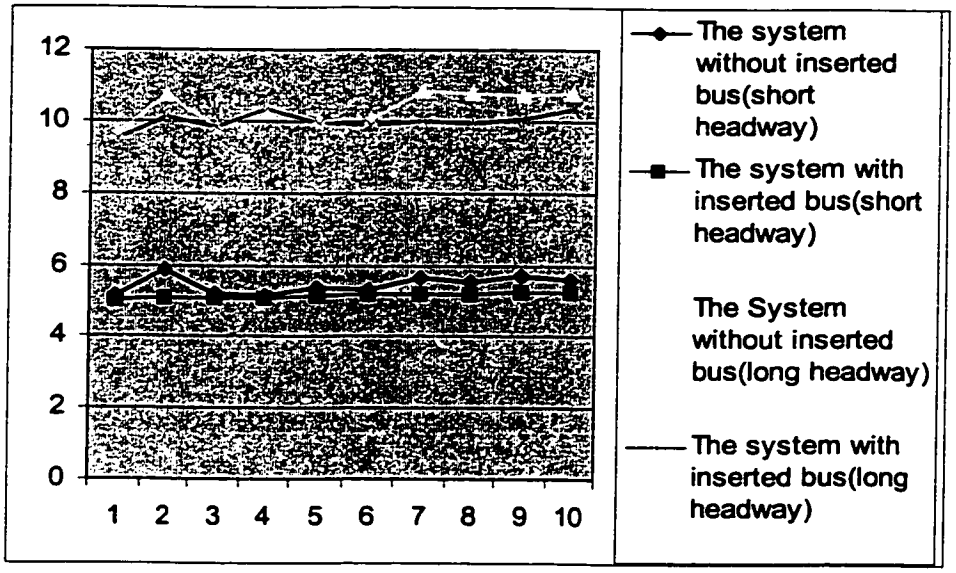


Figure 21-c: Actual average headway in the Schedule Control scenario

Charts in Figure 22 describe the change in actual average bus on-time percentage with and without inserting a bus into the system for the different scenarios. Figure22-a depicts the No Control scenario, Figure22-b the Headway Control scenario, and Figure22-c the Schedule Control scenario.

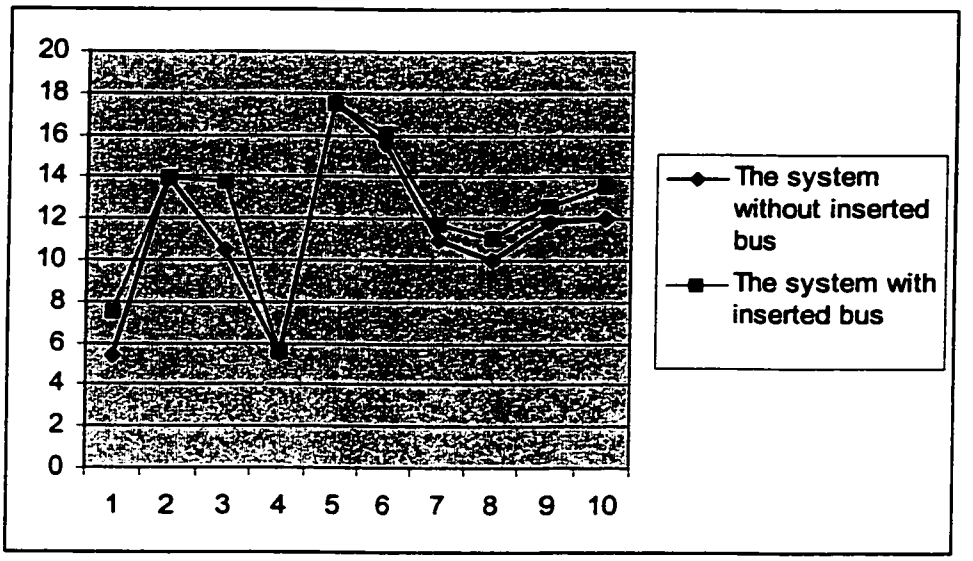


Figure 22-a: Average percentage of buses on time in the No Control scenario

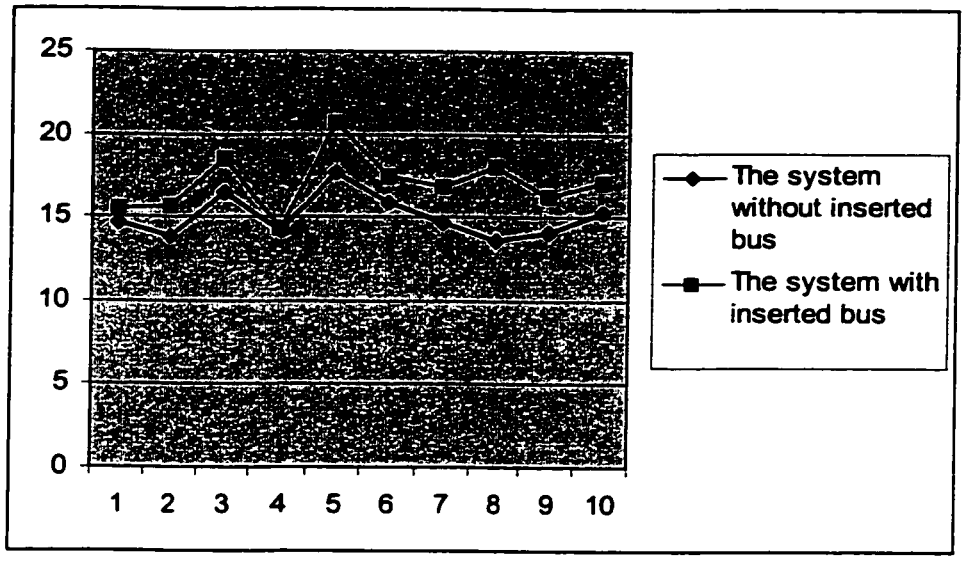


Figure 22-b: Average percentage of buses on time in the Headway Control scenario

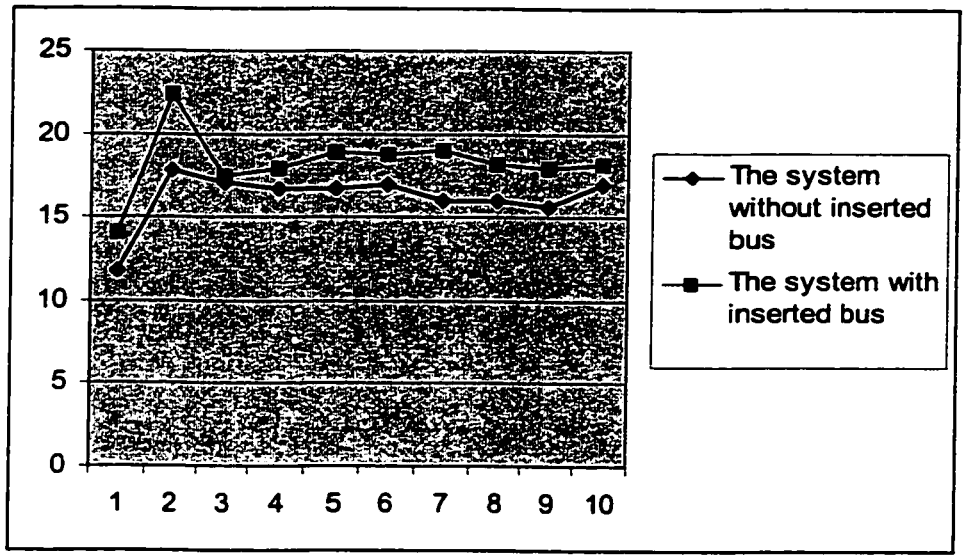


Figure 22-c: Average percentage of buses on time in the Schedule Control scenario

The value of the average headway between two buses is expected to be shorter after inserting one bus into the route. The value of the actual headway is based on the number of buses running on the route. Assume there are 5 buses running on the route in case one, and 6 buses running on the route in case two. Assuming buses are evenly spaced, it is not difficult to conclude that the headway in case one is larger than in case two. So, we expect that the average headway will be shorter after a bus has been inserted.

In our case, one trip is composed of a short headway section (with planned headway = 5 minutes) and a long headway section (with planned headway = 10 minutes). The planned travel time over the short headway section (from Baseline to Blair) is 37 minutes, and the planned travel time over the long headway section (from Blair to Orleans) is 17 minutes, so the planned trip travel time (from Baseline to Orleans) is 54 minutes. We also should consider the layover time at the end of each trip when we calculate the number of buses running on the route. The default minimum layover time is 4 minutes in the simulation model.

As the planned headway from Blair to Orleans is twice the planned headway from Baseline to Blair, every second bus leaving Baseline will stop at Blair. Let B_i be the i^{th} bus to leave from Baseline, and assume the odd numbered buses will go to Orleans and the even numbered buses will short turn at Blair. Assume B_1 leaves at time 0, and that the layover times are

planned to be as short as possible to minimize the total number of buses needed to cover the planned service over both directions. Then, B_1 is scheduled to arrive to Orleans at time 54, and is available to leave from Orleans at time 58 after a 4 minute layover. Hence, at the earliest, it is back to Baseline at time 58 + 54 = 112. Note that because of a minimum layover time of 4 minutes, B_1 can not leave Baseline for its third trip before time 116. However, to maintain a 5 minute headway at Baseline, the departure time from Baseline must be a multiple of 5: hence, B_1 can not leave Baseline before time 120. Thus, to cover the "long cycle" between Baseline and Orleans, 12 buses are needed.

Using a similar logic, we can determine how many buses are needed to cover the "short cycle" between Baseline and Blair. Bus B_2 will leave Baseline at time 5, and arrive at Blair at time 42. Given the layover time, the earliest time at which it can leave from Blair is 46. Note however that B_1 will leave Orleans at time 58, and given a 10 minute headway from Orleans, then buses must have left Orleans at time 48, 38 and 28, arriving at Blair 17 minutes later. Hence, "odd numbered" buses coming from Orleans will be leaving Blair at time 45, 55, and every ten minutes from then on. Hence, the even numbered buses must leave Blair at times which are multiples of 10, i.e. 30, 40, 50 and so on to maintain the 5 minute headway at Blair. Thus B_2 will leave Blair at time 50, after an 8 minute layover, arriving at Baseline at time 87. The earliest departure time for B_2 for its second trip from Baseline is therefore time 91. However, as buses leave at times which

are multiples of 5 to maintain the headway, B_2 would leave at time 95,or 90 minutes after the start time of its first trip. Hence, 9 buses are needed to cover the "short cycle" between Baseline and Blair, for a total of 21 buses for the two cycles.

If a breakdown occurs, the total number of buses will go down to 20. This will likely have a negative impact on the actual average headway. We can estimate the average headway following a breakdown, assuming no extra bus is introduced to replace the broken bus. We first present some notation and make some simplifying assumptions.

Let L_1, L_2, ..., L_{12} represent the sequence of buses operating on the long cycle between Baseline and Orleans, numbered according to the order in which they left Baseline for the first time; for convenience, we let $L_{13} \equiv L_1$. Similarly, let S_1, S_2, ..., S_9 be the sequence of buses operating on the short cycle between Baseline and Blair, numbered according to the order in which they left Baseline for the first time; for convenience, we let $S_{10} \equiv S_1$. Note that the planned headway between L_i and L_{i+1} is 10 minutes; similarly, the planned headway between S_j and S_{j+1} is 10 minutes. We assume that following a breakdown, all remaining buses can manage to adhere to their schedule. We need to consider two cases.

Case 1: The bus that breaks down is running the "short cycle".

It is easy to show that at any given point in time, 18 buses run between Baseline and Blair. To see this, note that there are 9 buses running the short cycle, and that each of these buses is followed by a bus running the long cycle. Without loss of generality, for an eastbound trip, we could have S_j, followed by L_i, followed by S_{j+1}. Note that when L_i reaches Blair, it goes on to Orleans, while S_j and S_{j+1} would start their westbound trip from Blair. However, some other bus, L_k, coming from Orleans, will insert itself between S_j and S_{j+1} for the westbound trip. Assume bus S_j breaks down. Following the breakdown, only 17 buses are left running the short cycle. Assume without loss of generality that S_j was preceded by L_1 and followed by L_2. As all buses maintain their schedule, the headway between successive buses will remain at 5 minutes, except between L_1 and L_2 where the headway will be 10 minutes. Hence, under this simplifying assumption, we can estimate the average headway between Blair and Baseline following a breakdown to be: (16 * 5 minutes + 1 * 10 minutes) / 17 buses, which equals 5.294 minutes. Hence, we can expect that inserting an extra bus following the breakdown would improve the average headway by 0.294 minutes between Blair and Baseline. Note that as S_j does not run between Blair and Orleans, the average headway between Blair and Orleans remains at 10 minutes under this simplifying analysis.

Case 2: The bus that breaks down is running the "long cycle"

Assume now that bus L_i breaks down. Note that L_i would have normally run across the whole route, and the full cycle takes 120 minutes (including 4 minutes of layover time in Orleans, and 8 minutes of layover time at Baseline). Over this full cycle and according to schedule, L_i would spend 38 minutes (including travel and layover times) between Blair and Orleans, and 82 minutes between Baseline and Blair. If L_i is no longer running, there exists a "gap" somewhere along the route. This gap moves around over time. Over a 120 minutes cycle, this gap would be between Baseline and Blair for 82 consecutive minutes, meaning that for this period of time, there would be only 17 buses running between these two stops: assuming, as before, that all buses maintain their schedule, the average headway would be 5.294, as calculated above. However, for 38 minutes out of the 120 minute cycle, the "gap" would be between Blair and Orleans, which means the average headway between Baseline and Blair would be 5 minutes. Hence, the average headway between Baseline and Blair may be estimated as follows:

$$82/120(5.294) + 38/120(5) = 5.2 \text{ minutes}$$

The average headway between Blair and Orleans may be calculated by realizing that only buses operating along the long cycle will visit this section of the route, and that following the breakdown of L_i, only 11 out of 12 buses are left running the long cycle. Hence, the estimation of the average

headway is given by: (10 * 10 minutes + 1 * 20 minutes) / 11 buses, which equals 10.91 minutes.

Note that in our small experiment summarized in Table 8, we do show some improvement in average headway following the insertion of an extra bus after a breakdown, as expected. The actual improvement reported does not necessarily equal the expected improvement as calculated above. This can be explained by a number of reasons. First, the above calculations are based on the simplifying assumption that all remaining buses would be able to maintain their schedule perfectly following the breakdown. In practice, even when there are no breakdown, buses can not be expected to perfectly adhere to their schedule as various conditions along the route will cause delays (traffic conditions, number of passengers at stops, traffic lights, etc...). A breakdown creates additional imbalances over the route, as described earlier in the thesis, so that both the average headway and the variance of the headway are expected to increase (this has been shown in Ji [5]). Second, in the various simulation runs reported, it is not known whether the bus which broke down was running the short cycle (case 1) or the long cycle (case 2). Finally, the time at which the breakdown took place during the simulation and the location of the bus when the breakdown occurred are important factors: for example, if the breakdown occurred minutes before the end of the simulation run, between Baseline and Blair, the impact of inserting a new bus on the headway between Blair and

Orleans may not be as significant as if the breakdown took place earlier in the simulation run.

Inserting a new bus into the route will bring the system back to normal. It also helps the buses which follow the broken to return bus back to their schedule. Thus, we expected the average trip time would be shorter, and the average percentage of buses on time would increase after having done the insertion. Our small experiment suggests that this is the case. In Figures 20-a, 20-b and 20-c, we see that the average trip times with an inserted bus are smaller than without an inserted bus for all simulation runs. This suggests that the average trip time becomes shorter after inserting a new bus into the route. Similarly, in Figures 22-a, 22-b and 22-c, the average percentage of buses on time in the system with an inserted bus is greater than without an inserted bus for all simulation runs, again suggesting system wide improvement as expected.

The goal of this validation is not to quantify the improvement associated with inserting a new bus following a breakdown, but rather to show that simulation results are affected by the introduction of the new bus, and that the changes in the results are "reasonable", or "in the expected direction". Tables 7, 8 and 9 demonstrate this. More research would be needed to determine if inserting a new bus following a breakdown indeed provides significant improvements to the service: the results of Ji [5] suggest that this is the case. Our work has made it easier for the user to insert a bus during the running of the simulation.

V-2-2-2. Validation from the run time bus progress screen:

We have another means to validate whether the new inserted bus is in fact actually running in the simulation system. It is watching the new inserted bus through the run time bus progress screen. Before doing the central control action, it is better to change the screen view to the run time bus progress screen. After having done the Insert Bus Following Breakdown Action, we expected that the new inserted bus would appear on the bus progress screen. In our experiment, we captured the new inserted bus from the bus progress screen after doing the Insert Bus Following Breakdown Action. Figure 23 is snagged in the experiment. The bus with busID 1111 is the new inserted bus.

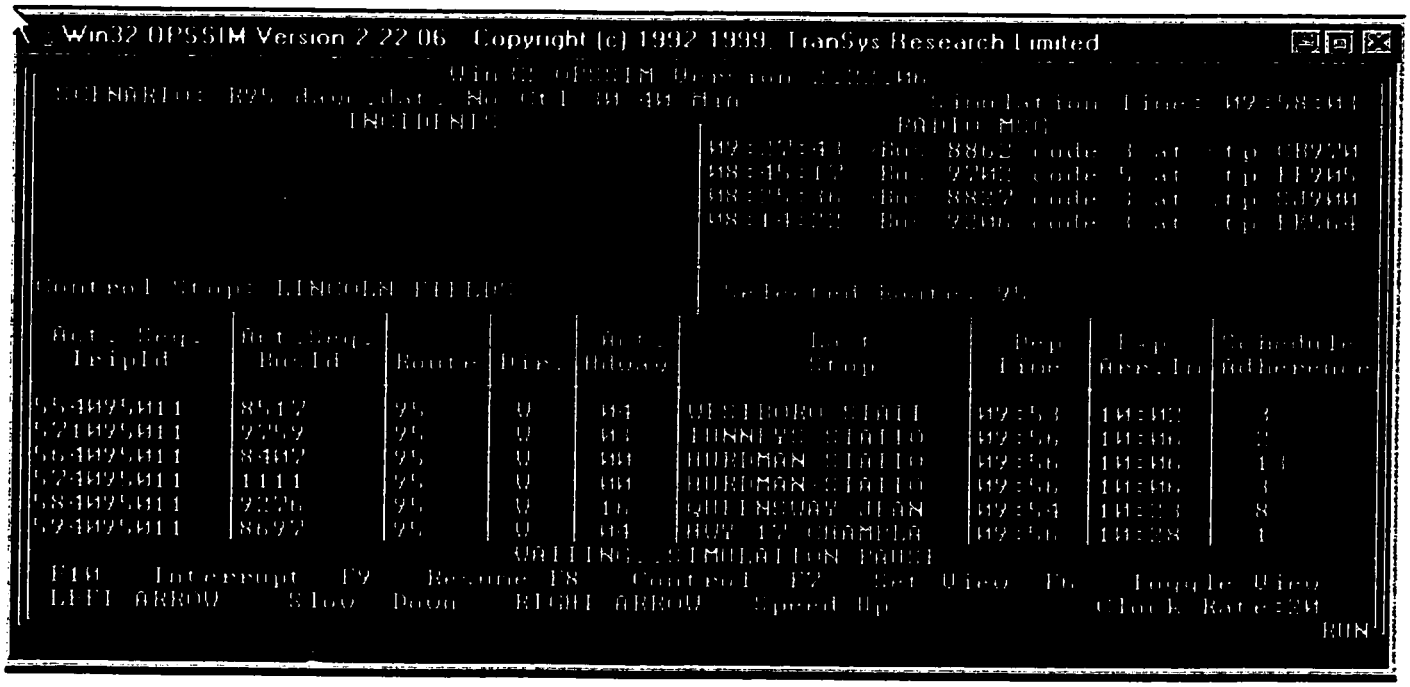


Figure 23: View of the new inserted bus

V-3. Tests on the New Run Time Bus Progress Screen:

As for CCAM in the previous section, the following tests have been conducted in order to evaluate the user-friendliness in entering data in the screens associated with the Bus Progress Screen. The following data shown in the Bus Progress Screen, namely the actual headway, the schedule adherence and the departure times from the "last stop" are generated by the core of the OPSSIM simulation model, not by the new bus progress screen module. Thus, the validation of these data falls outside of the scope of this thesis.

V-3-1 User-friendliness of BPS:

BPS1: Tests to make sure the toggle function (F6 key) works.

BPS2: Tests to make sure that the interface of the Set Control Stop Window works correctly and that the control stop selected by the user actually appears in the BPS.

BPS3: Tests to make sure the Set View window works correctly, and that the View selected by the user (i.e. either the Stop Statistics View, the Link Statistics View, the Stop and Link Statistics View or the Bus Progress Screen View) actually appears during the simulation (use of the F7 function key).

Again, the BPS passed all these tests successfully: the BPS provides a new, user-friendly way for controllers to view more relevant information during the simulation.

VI. SUMMARY AND FUTURE RESEARCH:

VI-1. The Enhanced Capabilities of OPSSIM simulation model:

A new Central Control Action Module, incorporated in the OPSSIM simulation model has the potential to allow users to insert control commands into the simulation during the run time. As a first step in this direction, we have designed and implemented one central control action, the Insert Bus Following Breakdown, in the Central Control Action Module. This allows researchers to do much broader investigations on real time control strategies dealing with various contingencies such as traffic accidents. As a training tool, the new enhanced feature makes the simulation more like the real operation platform.

In addition, the Run Time Bus Progress Screen allows the user to view the bus progress information at one chosen control point. It enables the OPSSIM simulation model to display more valuable information regarding bus location and bus performance vis-à-vis headway and schedule adherence.

The two new features provide new value to the OPSSIM simulation model and make it a more effective and efficient tool for researchers and central controllers.

VI-2. Future Research:

Future work dealing with the Central Control Actions Module includes:

- Implementing the Insert Trip control action.
- Implementing the Cancel Trips control action.
- Implementing the Short-turn control action.
- Doing more experiments with the OPSSIM simulation model using the various Central Control Actions to investigate which action is more efficient under various scenarios.

Future work in the enhancement of the OPSSIM simulation model includes:

- Designing and incorporating Driver Control Actions. Driver control actions should include "slow down" and "speed up", for buses running early or falling behind schedule respectively, as well as "bypass stop", and "leap-frogging". In developing these actions, the programmer should provide the user the means to define under which circumstances the various actions can be taken by the driver. This would be a first attempt at implementing a coordinated decentralized control system, a much more complex system to set up and manage, but a system which has its merits and a good potential for service improvement.
- Integrating the traffic light system into the OPSSIM simulation model.

- Implementing the various other enhancements in the research proposal of Transys Research Ltd listed in ChapterII-2.

From the "programming" perspective, it is also desirable to redesign the visual interface of the OPSSIM simulation model using visual modeling. The current user interfaces are all menu driven. While appropriate in the early 1990's, the menu interface can now be replaced by a more versatile and user-friendly "window interface". The use of object-oriented language, such as Visual C++, or Visual Basic, instead of the procedure language C is also suggested. These enhancements would facilitate bus insertion following a breakdown as the user could simply highlight the relevant information from the bus progress screen, such as the bus ID of broken bus, with a mouse, and click on an "Insert Bus Following Breakdown" icon which would trigger the action automatically.

The use of the OLE (Object Linking and Embedding) technology into the new designed OPSSIM simulation model should also be considered, to add route map and bus object into the user interface thus giving the user a view which more closely parallels the screens available in OC Transpo's control room.

Finally, research on Automatic Intelligent Control Centers with control actions and strategies done by a computer controlled system rather than through human intervention is very promising direction.

VI-3. Conclusion: The Challenges met in Our Work

To understand the structure and functionality of the OPSSIM simulation model rapidly without proper document was a great challenge for us when we started to work on this project.

The OPSSIM simulation model is very complex. To clearly understand the logical structure and each component in the system within a short time frame was not easily done. Furthermore, there was no documentation. Facing this challenge, we first did tests on the simulation model in order to understand each step of running the simulation and every data required or displayed by the simulation model. Then, we tried to understand the AVLC data system and the relationship between it and the OPSSIM simulation model. This step was critical. It helped us to design the Central Control Action Module.

When we started working on the Run Time Bus Progress Screen, we had to face two more challenges. One was how to get the value of the variables that would be displayed in the bus progress table. The other was how to integrate the bus progress screen code with the existing source code of the OPSSIM simulation model.

There are thousands of variables involved in the OPSSIM model. We needed to find the variables that hold the required data from the thousands of variables. We literally had to guess the meaning of variables from their names, and then, write a short program to display the data stored in the variables to see whether it was the required variable. In addition, we also

read some code in order to find how and when the OPSSIM program calculates the value of the variable. It is also very important for us to decide when we will cite the variable in our program.

The key of solving the second challenge was to find the connection point in the source code of OPSSIM simulation program that can make the link with the Run Time Bus Progress Screen program. After some research, we moved our focus on the CENTCTRL.C code file and DRIVCTRL.C code file. We read the two files line by line to try to find the logical relationship among those functions. After knowing how these functions work, it was not very difficult to find the connection point.

From beginning to end in this research project, we met lots of challenges. Whether these challenges were big or small, we addressed them to the best of our capabilities, given the resources and time available. The process of overcoming these challenges was very useful for us. It cultivated our capability and methodology to solve problems, which is invaluable for future research efforts, and in the workplace.

APPENDIX:

Appendix A: the list of variables which will be used in the Insert Trip(s) Following Breakdown module

blockList: the link list holding the block data of route 95 in the Block.dat file.

tripList: the link list holding the trip data of route 95 in the Trip.dat file.

tripList_restBk: the link list holding the trip data of the rest trips in the same block in which the bus broke down.

trippatList: the link list holding the trip pattern data of route 95 in the Trippat.dat file.

trippatvcList: the link list holding the trip pattern vector data of route 95 in the Trippatvc.dat file.

trippatvcList_Pat: the link list holding the trip pattern vector data of route 95 with given trip pattern.

blockId_Bk: the block identifier in which the bus broke down.

nodeEd_Bk: the end node at which the broken bus will terminate the running trip.

nodeEd: the end node in one given block.

timeEdSch_Bk: the scheduled end time at which the broken bus will terminate the block.

daytype_Bk: the day type under which the broken bus is running. It is the same as the trip type mentioned before.

runIdMax_Bk: the max runId number in one block with the given day type in the Block.dat file.

trippat_Bk: the trip pattern with which the broken bus is running.

lineId_Bk: the lineId in which the running trip of broken bus is.

dir_Bk: the direction at which the broken bus is running.

deltaTime: the delta time between the start node set by user and the end node at which the broken bus will terminate the running trip.

routePatIdMax_Bk: the max routePatId number of route 95 under the given direction in Trippat.dat file.

seqNoMax_Bk: the max seqNo. of route 95 under the given direction in trippat.dat file.

trip_New: a new trip data.

trippat_New: a new trip pattern data.

trippatvc_New: a new trip pattern vector data.

block_New: a new block data.

Appendix B: Description of the main functions in Bus Progress Screen Program.

Some primary functions are fundamental to the Bus Progress Screen program. These functions are described in detail below:

void Paint_Bus_Process(void){ }function:

This function is used to draw the table frame. Some functions in the OPSSIM library are invoked here, such as, OPS_scrn_text() function used to

write the string on the screen, OPS_scrn_char()function used to write a character on the screen. Some global constants are also used in the function, such as, CROSS expressing the signal +, VERTICAL_SINGLE expressing the signal |. These constants are all used for drawing the table on the screen.

STOPDES* stopdeslist(void){ } function:

This function creates the link list of stop name, which will be used in the Control Stop Selection Window to show the list of stop name and let the user choose one from them.

STOPDES* findstopD(STOPDES*, BYTE, INT2B){ }function:

This function searches the link list of stop name created by the above function, finds the stop name selected by the user, then, returns it. This function will be used to write the control stop name on the screen.

void Screen_Clear(INT2B){ }function:

This function is used to clear the screen before writing the data on the screen.

void Bus_Display(BUSSRN*){ }function:

The function writes all data into the bus progress table. It will refresh the screen when one of the buses listed on the table leaves the last stop.

void CtrStop_Display(void){ }function:

This function will build the Control Stop Selection Window. This window includes a control stop name list, the selected(X) or not selected(.) mark, and a cursor(➔). The UpArrow key or DownArrow key are used to move the cursor up and down.

The number of the control stop names in the list is greater than the capacity of the window on which the list is displayed. Thus, a window scroll is implemented here. Up to twelve stop names can be shown on the window. If the cursor is on the last line of the window, and there are stop names in the list that are not displayed on screen, press the DownArrow key, then, the window will scroll down one line. A new stop name will be displayed on the bottom line. The window can also scroll up.

Besides enabling the window scroll to display all stop names in the list, another important task of this function is to display whether the stop name is selected or not to the user and mark the last choice of the user with the symbol "X". By default, no mark will be displayed at the left of the stop name. If the user moves to cursor to the stop name that he wishes to select, and presses the Return button, a mark(X) will appear on the left of the stop name indicating that the stop has been selected. Only one stop will be so marked, and the function always indicates the user's last choice. Once a new selection is made by the user, the old selection will be dropped off. If the user reopens the Control Stop Selection Window before the simulation terminates, the mark will still be at the last selected stop name. This helps users remember which stop was selected at the last time.

In order to build a more user friendly window, we designed the selected mark to move based on the user's decision. This feature is very convenient for user to change the decision, but only one selected mark is allowed in the window because we only allow the user to monitor one control stop at a time.

void Bus_Progress(BYTE, INT2B, STOPDES *){ } function:

This function has a very important role in the Bus Progress Screen program. It is responsible for implementing the algorithm of looking for the sequence of buses which will arrive at the control stop and displaying them on the run time screen.

The procedure of the function is given below:

Step1: Find the control stop, and display the name of control stop on the screen.

Step2: Find the sequence of buses which will arrive at the control stop in order. This is done by reading the bus arrival queue generated by the simulation model at the control stop. If there are currently no buses between the selected control stop and the terminal upstream, the table will remain empty. Go to step4.

Step3: Call Bus_Display() function to display the sequence of buses onto the screen.

Step4: Stop.

Appendix C: The Data Types will be used in Bus Progress Screen Module

The following data structures are provided to assist future program developers in their efforts to introduce new features and enhancements to the simulation model. In implementing the Bus Progress Screen, there are three kinds of data type in the source code of previous version of OPSSIM simulation model used to get the required information that will be shown on the screen. They are **BUSQUEUE**, **STOP**, and **BUS**.

BUSQUEUE: BUSQUEUE is the defined data type. It takes the following form:

```
typedef struct   tagBUSQUEUE
{
        INT2B                   nRoute ;
        INT2B                   nBusId ;
        INT4B                   iTime ;
        struct tagBUSQUEUE      *psPrior ;
        struct tagBUSQUEUE      *psSucc ;
 } BUSQUEUE ;
```

where:

INT2B: long integer. (C standard data type)

STOP: STOP is the user defined data type. It is very complex and includes more information about the stop. It is defined as the following:

```
typedef struct   tagSTOP
{
        INT4B           iNodeId ;
        BYTE            cBsiId[ BSI_SIZE ] ;
        INT2B           nStnLookup ;
        INT2B           nDir ;
        BYTE            cDescription[STOP_DESCRIPTION ] ;
        FLOAT           fRejection ;
```

```
        FLOAT           fQueueReject ;
        FLOAT           fQueueLoad ;
        FLOAT           fArrivalSigma ;
        ARRIVALS        sMeans[ MAX_ARRIVAL_MEANS ] ;
        FLOAT           fUnloadPercent[ 3 ] ;
        BOOL            bRunDisplay[ 2 ] ;
        BOOL            bArrivalModify ;
        FLOAT           fArrivalModifiers[MAXSTOPFIELDS ] ;
        BOOL            bRegulationModify ;
        FLOAT           fRegulationModifiers[ 6 ] ;
        BUSARRIVALS     sBusArrivals[ ARRIVALBUSES ];
        INT2B           nStatus ;
        INT4B           iOpenTime ;
        BUSQUEUE        * psBusQueue ;
        INT2B           nQueue ;
        INT2B           nStats_Obs ;
        STATS           sStats_Obs[ STATISTIC_SAMPLES] ;
        INT2B           nBuses15 ;
        INT2B           nBusesDay ;
        INT2B           nObsDay ;
        FLOAT           fHeadwayDay ;
        FLOAT           fHeadwayDaySq ;
        INT4B           iPassengerDay ;
        INT4B           iPassengerHead ;
        INT2B           nDayQueue[ MAX_DAY_QUEUES ] ;
        INT2B           nDayLoads[ MAX_DAY_LOADS ] ;
        INT2B           nDayHeadway[ MAX_DAY_HEADWAY] ;
        INT2B           nDaySch[ MAX_DAY_SCH ] ;
        STOPDATA        *psStopData ;
        BASESTOP        *psBase ;
        struct tagSTOP  *psPrior ;
        struct tagSTOP  *psSucc ;
} STOP ;
```

where:

INT2B: long integer. (C standard data type)

INT4B: integer. (C standard data type)

FLOAT: float. (C standard data type)

BOOL: boolean. (C standard data type)

BYTE: char. (C standard data type)

The other: the user defined data type.

BUS: the user defined data type. It includes more information about the bus. It is defined as the following:

```
typedef struct   tagBUS
{
        INT2B           nBusId ;
        BUSTYPE         *psBusType ;
        struct tagBUS   *psPrior ;
        struct tagBUS   *psSucc ;
        INT4B           iActualPower ;
        INT2B           nPassengerLoad ;
        INT4B           iLoadPercent ;
        #if defined(AVLC)
        TD_TRIP         *psRunTrip ;
        TD_TRIPPATTERN    *psRunPattern ;
        TD_TRIPPATVEC     *psRunTripPatVec ;
        #endif
        #if defined(APC)
        APC_Run         *psRun;
        APC_Trip        *psRunTrip;
        APC_Pattern     *psRunPattern;
        APC_Vector      *psRunTripPatVec;
        #endif
        LINK            *psRunLink;
        INT2B           nRoute;
        INT2B           nDir;
        BOOL            bStartTrip ;
        BOOL            bContinueTrip ;
        STOP            *psPriorStop ;
        STOP            *psNextStop ;
        INT4B           iLastProbTime ;
        BOOL            bBreakDown ;
```

```
        INT2B           nCodeBreak ;
        STOP            *psBreakStop ;
        BOOL            bShortTurn ;
        INT4B           iActualStartTm ;
        INT4B           iSchPriorStopTm ;
        INT4B           iSchPriorNodeTm ;
        INT4B           iSchNextNodeTm ;
        INT4B           iSchNextStopTm ;
        INT4B           iFinishUnload ;
        BOOL            bByPassStop ;
        INT2B           nRemainStops ;
        INT2B           nRemainLowStops ;
        FLOAT           fHighStopsPer ;
        FLOAT           fRemainHighStopsPer ;
        INT4B           iBusStop[ 11 ] ;
        FLOAT           fArrivalRate ;
        INT4B           iLastNode ;
        INT4B           iLastTime ;
        INT4B           iStartUnloadTime ;
        INT4B           iPassengerTime ;
        #if defined(TRAFFIC)
        CORRIDOR        * psCor;
        INTERSECTION    * psIntersection ;
        #endif
        INT4B           iTransitTime ;
        INT4B           iTransitTimeLeft ;
        INT4B           iTransitDelay ;
        #if defined(TRAFFIC)
        PLATOON         *psPlatoon ;
        #endif
        INT4B           iHeadway;
        INT4B           iLastStopDepTime;
        INT4B           iSchAdherence;
} BUS ;
```

where:

INT2B: long integer. (C standard data type)

INT4B: integer. (C standard data type)

FLOAT: float. (C standard data type)

BOOL: boolean. (C standard data type)

BYTE: char. (C standard data type)

The other: the user defined data type.

These data types are defined in the structure.h header file of OPSSIM simulation model. During the simulation running time, the simulation will create these data types, and save the value in each data type.

REFERENCE:

1. CIGGT, "OC Transpo Transit Simulator OPSSIM version1.3: User Manual", August, 1993 version.
2. CIGGT, the source code of OPSSIM simulation model, 1993.
3. H.S. Levinson, "15 Synthesis of Transit Practice: Supervision Strategies for Improved Reliability of Bus Route ", Transportation Research Board, National Research Council, Washington, D.C., September, 1991.
4. Lule Chen, "Study of Bus Control Strategies By Computer Simulation", Master's thesis, University of Ottawa, 1994.
5. Ji, Xu, "Study of Dynamic Headway Control Bus Dispatching Rules", Master's thesis, University of Ottawa, 1995.
6. Guey-Shii Lin, Ping Liang, Paul Schonfeld, and Robert Larson, "Adaptive Control of Transit Operations", http://www.fta.dot.gov/library/technology/APTS/ITS/COVER.HTM November, 1995.
7. Robert F. Casey, Lawrence N. Labell, Ross Holmstrom, and Joseph A. LoVecchio, "Advanced Public Transportation Systems: The State of the Art Update'96 " , January, 1996.
8. Robert Chapleau, Bruno Allard, and Martin Trepanier, "Transit Path Calculation Supported by Special Geographic Information System-Transit Information System", Transportation Research Report No.1521,1996.
9. David B.Roden, "Forecasting Travel Time", Transportation Research Report No.1518, 1996.