AN EXPERIMENTAL INVESTIGATION OF DYNAMICALLY RECONFIGURABLE COMPUTER NETWORK ARCHITECTURES THROUGH SIMULATION

by

Anjali Venkateshwaran

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

APPROVED:

Richard E. Nance, Chairman

Osman Balci

Dennis G. Kafura

January 4, 1988

Blacksburg, Virginia

AN EXPERIMENTAL INVESTIGATION OF DYNAMICALLY RECONFIGURABLE COMPUTER NETWORK ARCHITECTURES THROUGH SIMULATION

by

Anjali Venkateshwaran

Richard E. Nance, Chairman

Computer Science

(ABSTRACT)

The research described in this thesis is divided conveniently into three components: (1) the credibility assessment of a simulation model for the investigation of dynamically reconfigurable computer network architectures, (2) a comparative study of the standardized time series method of simulation output analysis, and (3) an experimental comparison of the effects of dynamic reconfigurability on message transmission delays and network throughput.

The credibility assessment relies almost completely on verification procedures applied to both communicative and program representations of the model. In the absence of an extant system, validation consists of extensive program traces to assure that model behavior matches expectations and reflects no inconsistencies.

Application of a standardized time series technique produces the advantages reported by other researchers with regard to sampling efficiency (information derived per sample unit) when dynamic reconfigurability is precluded. The inherent non-stationarity induced by reconfiguration reveals the sensitivity of standardized time series and the consequent adjustment to preserve coverage. A compromise between coverage and sampling efficiency prompts the choice of the batch means method for experimental comparison.

Experimental comparison shows that under high traffic variability reconfigurability produces reductions in mean message transmission delay that range from five percent for a six-node network to a factor of eleven for a sixteen-node network. Limited performance analysis underscores the importance of minimizing reconfiguration period length. Mean network delay appears to exhibit sensitivity to an interactive effect between reconfiguration period length and the level of message demand.

Acknowledgements

I wish to thank Professor Richard E. Nance for his guidance and valuable suggestions throughout the course of this research. I am grateful for his support and assistance which made my work a lot more enjoyable.

I also wish to thank Dr. Osman Balci and Bob (Dr. Robert L. Moose) for their helpful suggestions. I am grateful to the members of the Computer Science Department for their support, especially for prompt replenishment of my computer account funds and for her efforts in procuring the Simula compiler. I am also grateful to of the Systems Research Center for all her help. I wish to thank Dr. Dennis G. Kafura for serving on my committee.

Finally, I would like to thank my family: my parents , my parents-in-law , my brother and his family, my other brother , my sister and her family for their continued support and encouragement. I am indebted to my husband for his help, encouragement, support and understanding at all times. To him, I dedicate this manuscript.

Table of Contents

List of Illustrations

List of Tables

1.0 INTRODUCTION

The dynamic hierarchical architecture, first conceptualized by Nance [Nance 1979] for Naval applications, also meets the requirements of specialized areas of network applications such as the nuclear power industry. The architecture is identified as offering advantages of survivability and reliability beyond those of more traditional network architectures [Derrick and Nance 1986]. Dynamic reconfigurability may exact a cost in terms of other performance criteria (message transmission times and throughput) but the degree of cost is unknown.

The dynamic hierarchy is defined by Moose [Moose 1983]:

> *The dynamic hierarchy is a generalization of the centralized, tree network. It shares with its conventional counterpart the characteristic that, at any given time, a single node resides at the logical apex of the hierarchy. The remaining nodes are configured in a tree structured topology. If, in addition, we allow the apex to vary, the result is a dynamic hierarchical network.*

The concept permits variations in the sets of active and inactive links. Active links have nonzero traffic rates while inactive links carry no traffic. The set of active links at all times conform to a tree topology [Moose and Nance 1987]. The dynamic hierarchy also permits variations in the link capacity assignments to improve the overall network performance.

1.1 Research Problem

The research leading to this thesis centers on the following objectives:

1. Verification of the simulation model of the dynamic hierarchy local area network, (DHLAN). Model verification and validation are necessary for a successful simulation study. Since a model is an abstraction of the problem under study, it is important to

establish the model's credibility, and verify that the model possesses a sufficiently accurate representation with respect to the study objectives.

2. A comparison of three statistical methods for analyzing simulation results:
   - batch means,
   - replications, and
   - standardized time series.

   Standardized time series is a relatively new technique for analyzing simulation output data, suggested by Schruben [Schruben 1983]. The complexity of the dynamic hierarchy network model and the applicability of this technique to the model, provides a good opportunity to compare it with the popular statistical methods of replications and batch means.

3. Comparison of a dynamically reconfigurable network architecture with its static counterpart. The dynamic hierarchy, proposed as more appropriate for specialized areas of network applications, incurs a cost. A comparative cost/benefits analysis of representative local area networks (LANs) is essential to an initial assessment of the dynamic reconfigurability concept.

4. Study of the delay-throughput characteristics for selected networks. A performance study of the network using a well designed experiment significantly contributes toward assessment of the credibility and acceptability of the simulation results. The simulation results provide useful insights into network behavior under the apex transition protocol, and help to "tune" key network parameters for further experimental work.

1.2 Thesis Organization

The second chapter gives an insight into the verification and validation of simulation models. Different statistical techniques for simulation output data analysis are described and simulation models of LANs and distributed systems are reviewed.

The third chapter contains the structured verification of the dynamic hierarchy simulation model. It describes the network characteristics and its behavior during apex transition, and gives an overview of the important network components of the simulation model. Program traces to verify the correctness of the DHLAN simulation model and the correct application of the three techniques, namely batch means, replications and standardized time series for comparison purposes, are included in this chapter.

The fourth chapter discusses the design of experiments to achieve the objectives mentioned above. Issues such as random number generation, data independence, normality, steady state analysis, traffic matrix determination, link capacity assignment, packet arrival rate/node determination and confidence interval estimation for the network parameters are included in this chapter.

The fifth chapter contains the results obtained for the different experiments performed and a discussion of the experimental results. The final chapter emphasizes conclusions resulting from this study and recommendations for future research.

2.0 LITERATURE REVIEW

This chapter summarizes the results of previous work done in the areas related to this research. The first section deals with issues relating model verification and validation; the second section discusses the important statistical techniques for confidence interval estimation; and the third section reviews other simulation models of local area networks (LANs) and distributed systems.

2.1 Model Verification and Validation

2.1.1 Model Verification

Verification is defined as "in general, the demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development cycle" [Adrian et al. 1982, p. 188]. The life cycle of a model can be characterized by three chronological phases: problem definition, model development, and decision support [Nance 1983]. Model development can be further divided into six phases as depicted in Figure 2.1. The oval symbols in the figure represent the six phases, the dashed arrows depict the input to output translation processes, and the solid arrows represent the credibility assessment stages. The six phases are discussed briefly in the following paragraph.

The model development phase begins with the identification of the system definition and the objectives for the system study. Based on the system and objectives definition, a modeler formulates a conceptual model in his mind. Nance [Nance 1981] has developed a Conical Methodology which provides a structured approach for building models. The conceptual model is translated into a communicative model, which is defined as "a model representation

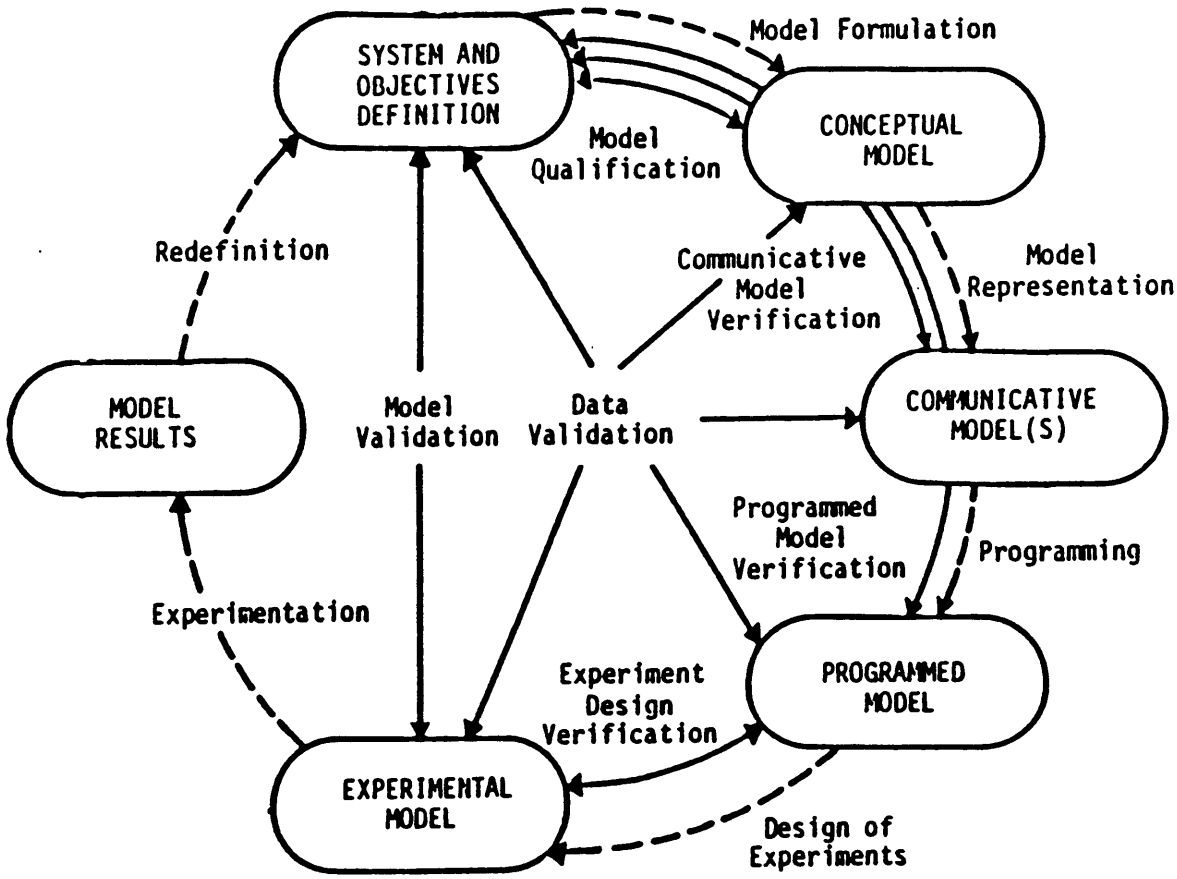


Figure 2.1. Model Development Life Cycle for a Simulation Study.: [Balci 1985, p. 8]

which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human" [Nance 1981, p. 14]. Flowcharts, pseudocode, block and logic diagrams are but some of the ways of representing a communicative model. The communicated model is then translated into a programmed model. A programmed model is "a model representation that admits execution by a computer to produce results" [Nance 1981, p. 14]. The programmed model is incorporated with an executable description of the experiment design to produce the desired information.

Substantiating the adequacy of transformation of an input phase into an output phase constitutes model verification. From Figure 2.1, clearly verification exists at all stages of model development. Since a model is an abstraction of reality, assessment of the credibility of each process in the development life cycle is crucial. The system definition and the objectives of the study should be verified periodically as both can change over time. All the assumptions made in abstracting from reality in defining the model should be explicitly specified and justified.

One must ensure that the conceptual model provides a sufficiently accurate representation of the system with respect to the study objectives. This process is termed as model qualification. A communicative model should be verified by "confirming it's adequacy to provide an acceptable level of agreement for the domain of intended application" where domain of intended application is "the prescribed conditions for which the model is intended to match the system under study" [Balci 1985, p. 9]. Techniques such as desk checking, model review, and graph-based analysis are used for communicative model verification.

"Substantiating that the programmed model represents the communicative model (and the system under study) within specified limits of accuracy" constitutes programmed model verification [Balci 1985, p. 10]. A programmed model that forms a basis for extensive experimental study should undergo a careful detailed structured verification. It is essential to verify that the programmed model conforms to the model specifications with sufficient accuracy. In addition to the techniques used for communicative model verification, instrumentation-based testing and functional testing can also be used to verify programmed models. The de-

sign of experiments should be verified by "substantiating that the experiments are correctly designed and their translation into an executable code is correctly done" [Balci 1985, p. 10]. Experiment design verification of the model entails answers to questions such as:

- Are appropriate statistical techniques implemented to define and analyze the simulation experiments?
- Is the initial transient phase problem adequately addressed?

[Balci 1986, p. 14].

2.1.2 Model Validation

Model Validation is substantiating that the experimental model within its domain of applicability, behaves with satisfactory accuracy consistent with the study objectives [Balci 1985, p. 10]. To gain confidence in the transformation process, a comparison of the simulator output and the actual process using identical input should not reveal significant differences. However, Balci [Balci 1987, p. 8] states that model validity is a necessary but not a sufficient condition for the credibility of simulation results.

The validation process depends upon the nature of the problem. The total spectrum of the problem of interest may be very broad. For example, consider a simple simulation model of a traffic intersection. The model may assume that all vehicles are of the same size, the intersection is accident-free, infinite queues are allowed, and all vehicles follow the traffic rules. In reality, the system is more complex than its simulation model and it is difficult, if not impossible, to develop a model that is sufficiently detailed to be accepted for any possible objective. One can improve face validity of the simple model by adding the effect on the intersection imposed by adjacent intersections and allowing only a finite number of vehicles to wait in a lane. However, despite its limitations, the model can be used to investigate certain defined objectives without arguing about the validity of the model as a general representation of the system.

A large number of models have been built to simulate the behavior of a non-existent system. When the actual process is specified by the modeler, validation can be accomplished using subjective techniques. However, in such cases subjective techniques may or may not be the most preferred choice for model validation. Simulation model validation generally falls into two broad areas: subjective validation techniques and statistical validation techniques. The statistical validation techniques are applicable only when the system being modeled is completely observable while subjective validation techniques can be used when the system being modeled is completely observable, partially observable or completely unobservable.

A few examples of subjective validation techniques are event validation, face validation and historical methods. Factor analysis, regression analysis, t-test and Theil's inequality are several examples of statistical tests for validation [Balci 1986]. Using the Standardized time series approach introduced by Schruben [Schruben 1983], Chen and Sargent [Chen and Sargent 1987] have developed confidence interval estimators for the difference between the means of two independent, strictly stationary phi-mixing stochastic processes to validate steady state simulation models of observable systems. However, all validation tests make some assumptions about the nature of a process. Thus the tests themselves are subject to questions of validity [Van Horn 1971].

2.1.3 Summary

In summary, verification is required at all stages of model development. Model validation requires testing that the model has been built "right;" in other words, validation testing establishes that the transformed model behavior adequately represents the real system consistent with study objectives.

The dynamic hierarchy network model is an abstraction of a proposed architecture specified by the project team. Due to the absence of a real system, concentration on the degree of confirmation of the model is very important. Hence, greater emphasis is placed on model verification than model validation. Verification resides in the assurance that the as-

sumptions and hypotheses underlying the DHLAN have been rightly formulated and the programmed model and the design of experiments have been correctly implemented. A structured review of the network using the verification techniques of desk checking, trace checking of event occurrences and instrumented-based testing facilitates model verification.

Validation is partly accommodated by verifying the model representation(s) produced during each stage of the model development life cycle. The network model is validated by determining the correctness of the program representation and by establishing that, under varying conditions, the model behavior conforms to the expected predictions.

2.2 Statistical Techniques for Confidence Interval Estimation

A large number of statistical methods have been developed over the years to analyze experimental observations. However, most of them assume that the observations are mutually independent and identically distributed, assumptions generally not warranted for simulation results. A single simulation run produces many observed values of a variable, but the value observed at one time is likely to be influenced by the value at some earlier time (the autocorrelation problem). The analysis literature identifies several approaches for obtaining independent and identically distributed observations from a simulation model. These approaches seek to control bias in measures of both mean and variance and ensure that the statistical estimates are consistent. The most prominent techniques are: replications [Conway 1963, Law 1977]; batch means [Conway 1963; Law 1977; Gordon 1978]; regenerative processes [Crane 1975; Fishman 1973, 1974]; standardized time series analysis [Schruben 1983; Goldsman and Schruben 1984]; and autoregressive processes [Anderson and Sclove 1978; Fishman 1978].

2.2.1 *Method of Replication*

One way of obtaining independent results is to repeat the simulation with a different random number stream. The method of replication is defined as making k independent simulation runs of length m observations for a total of N=km observations. Repeating the experiment with different random numbers for each of k samples of size m gives a set of independent determinations of the sample mean $\overline{X}(m)$. Combining the results of k independent measurements gives the estimates for the steady state mean μ and process variance σ^2 .

Two possible sources of error arise in using the replication method to construct a confidence interval for a steady state mean μ : (1) $E[\overline{X}(m)] \neq \mu$ (point estimator bias) and (2) the nonnormality of the sample mean estimates $\overline{X}(m)$. The normality of $\overline{X}(m)$ can be achieved by having a large sample size m. Due to the effect of the initial conditions $E[\overline{X}(m)]$ is a biased estimate of μ. The effects of the initial conditions can be removed or reduced by several approaches. The system can be started in a more representative state than the empty state or early values in the series can be ignored. The difficulty in determining a more representative state arises from the lack of theoretical knowledge of the system; hence the second approach is more commonly used to remove bias. The run is started from an idle state and stopped after a certain period of time. The entities existing in the system at that point of time are left as they are. The statistics gathered till that point in time are deleted, and the run is restarted with statistics being gathered from the point of restart [Conway 1963].

A drawback to eliminating the early simulation values is that the variance estimate needed to establish confidence limits must be based on less information. The reduction in bias is, therefore, obtained at the price of increasing the variance estimate and consequently the interval size. Law conducted experiments with the M/M/1 system and reported that if the number of replications in a simulation run is increased at the cost of reducing the individual run lengths, the estimate of the mean demonstrated greater bias, as a result of the initial

empty state. Hence a small number of repetitions is desired, bearing in mind the need for approximating the normal distribution with the sample means [Law 1977].

2.2.2 *Batch Means*

Another approach to establishing confidence intervals for simulation results is the batch means method. Batch means uses a single long run, preferably with the initial bias removed. The simulation run is divided into a number of segments to separate the measurements into batches of equal size. The final state of one run becomes the initial conditions for the next. This way of repeating a run is preferable to starting each run from an initial state, since the state at the end of a batch is a more reasonable initial state than the idle state. However, batch interdependence induced by object transition and process state transition across batch boundaries introduces correlation between the $\overline{X}(m)$. Separating the batches by intervals in which measurements are discarded is one way of reducing correlation, albeit with a loss of useful information.

Conway demonstrated that the variance to be obtained by using all the data and accepting the correlation between batches is less than that obtained from the reduced amount of data obtained by separating the batches. Hence, it seems preferable to include adjoining batches [Conway 1963]. A sufficiently long batch length is another way of reducing the effects of correlation. As the separation between the data increases, the effect of one piece of data on the value of the following data diminishes, and beyond some interval size it may reasonably be ignored. Hence, if the batch size m is sufficiently greater than this interval, the batch means may be treated as independent. A large k ($>$ 30) would achieve the normality of the $\overline{X}(m)$'s bringing the central limit theorem into effect.

The advantage of batch means over replications resides in the elimination of the initial bias except at the start of the run. However, like the replication method, a trade-off exists between the batch size and number of batches. Since the number of batches corresponds to the number of samples of an assumed normal distribution, this number should be a reasonable

limit to meet the assumption, and the batch size should be maximized in order to reduce the correlation between batches [Gordon 1978].

The experiments reported in Law's paper tested the batch means method against the replication method for many combinations of m and k. The results showed that in almost all cases, the batch means method was superior to the replication method and that the difference was statistically significant. However, it should be noted, that the results are for simple systems like the M/M/1 queue and (s, S) inventory system [Law 1977].

2.2.3 *Regenerative Techniques*

A system with the property that a set of values continues to recur at certain random intervals of time is said to be regenerative. The times at which a particular set of values chosen as a reference recurs are called regeneration points. Crane and Iglehart [Crane and Iglehart 1975] and Fishman [Fishman 1973, 1974] developed this method simultaneously. The notation of the former authors is used for describing this technique.

Suppose a discrete event simulation can be represented as a stochastic process $\{X(t) : t \geq 0\}$ with state space R^k, k-dimensional Euclidean space. A basic assumption of this method is that there exists an increasing sequence of random times $0 \leq b_1 < b_2 \ldots < b_n$, called regeneration times, such that at each of these times the process $\{X(t) : t \geq 0\}$ starts from a predefined state according to the same probabilistic structure governing it at time b_1. Furthermore, it also assumes that the expected time between successive regeneration times is finite. That is, between any two consecutive regeneration times b_n and b_{n+1}, say, the portion $\{X(t) : b_n \leq t < b_{n+1}\}$ of the regenerative process is an i.i.d. replica of the portion between any other two consecutive regeneration times. However, the portion of the process between 0 and b_1 , while independent of the rest of the process can have a different distribution [Crane and Iglehart 1975].

A regenerative model satisfies the Markovian property. At least one state exists in which future behavior is independent of the past behavior. Zero stock level, idle server and N cus-

tomers in the system are examples of potential regeneration points. A major drawback of this procedure for confidence interval estimation is the requirement that statistics be gathered over a fixed number of regeneration times so that the actual run length of the simulation is not known in advance. Also, regeneration states might be difficult to identify and can occur very rarely.

2.2.4 *Standardized Time Series*

The observations of a stationary stochastic process can be transformed into a standardized time series. The three methods formerly described above namely replication, batch means and regeneration processes are based on the principle of getting (or closely approximating) independent samples. Autocorrelation contains useful information. The time series approach accepts the presence of autocorrelation and makes it possible to compensate for its presence. A time series may be standardized by centering the series to have a mean equal to zero and dividing by a magnitude scaling constant. Virtually all time series are recorded discretely in time. The data may be organized following the method of replications or batch means [Schruben 1983].

Schruben [Schruben 1983] proposed four alternative interval estimators using a standardized simulation output series. Goldsman and Schruben [Goldsman and Schruben 1984] compared the classical confidence interval estimator with the four alternative interval estimators for stationary processes. The new estimators showed asymptotic properties that strictly dominated the classical estimator with data obtained from independent simulation replications or batches from a single simulation run.

The standardized time series retains all of the information in the original time series. For a given scaling constant, the original time series can be recovered from the sample mean and standardized time series, in contrast to the simulation methods of replication and batched means where the only information kept about a replication or batch is the series average. A relatively small number of replications or batches and a large sample size are reasonable for

data collection. The alternative interval estimators may be used with only one unbatched run of a simulation program. The confidence intervals have a tendency to compensate for the presence of simulation initialization bias, and their estimation requires very mild theoretical requirements for validity. One drawback of this method is that a considerable amount of calculation is involved, and it requires the availability of all data simultaneously, unlike the replication and batch means approach, which are able to summarize results as the run proceeds [Schruben 1983].

2.2.5 *Autoregressive Processes*

Autoregressive processes are a subset of stochastic processes. They are also termed stochastic difference equations. Their structure makes the evaluation of autocorrelation properties relatively simpler to calculate. The values of an autoregressive variable X_t are defined by the equation

$$X_t = -b_1 X_{t-1} - b_2 X_{t-2} - \ldots\ldots - b_p X_p + \varepsilon_t \qquad [2.1]$$

for t = p+1,... where ε_t is a random variable (usually assumed to be normally distributed) and the following conditions hold.

$$\begin{gathered} \mu = E(X_t), \quad E(\varepsilon_t) = 0, \\ E(\varepsilon_t^2) = \sigma^2, \quad E(\varepsilon_s, X_t) = 0, s > t \\ E(\varepsilon_s, \varepsilon_t) = 0, s \neq t \end{gathered} \qquad [2.2]$$

The process is determined by p coefficients b_i (i=1,...,p), together with the first p-1 values of X_t, which must be given as initial conditions [Anderson and Sclove 1978].

To estimate the variance of the mean μ, this method relies on the assumption that one can take a sequence of dependent observations $X_1, X_2, \ldots X_n$ and transform it into a sequence of i.i.d observations. It gives a concise representation for the variance of the mean μ. A suf-

ficiently large sample size is recommended for data collection. The autoregressive method offers several attractions:

(1) facilitates estimation of the distance between roughly uncorrelated observations.

(2) enables one to assess the extent to which bias due to initial conditions affects the sample mean.

Note that large data storage requirements can complicate analysis of the raw data; however, this can be avoided by forming batch means, say, of q observations and using these as input, [Fishman 1978].

2.2.6 *Summary*

In summary, the method of replication requires k independent simulation runs each of length m observations (n = km) and the method of batch means requires one long run of length n which is then divided into k batches each of length m observations (n = km). The standardized time series method has the advantages that confidence interval estimation tends to compensate for the initial condition bias and requires very mild theoretical requirements for validity. A major drawback of the regenerative method is the difficulty in identifying regeneration points. Autoregressive processes require that several assumptions be met regarding interactions among data values.

The regenerative technique is less suitable for the DHLAN because of the difficulty in identifying practical regeneration points. Also, the regeneration points may not reflect the true nature of the model's behavior due to initial condition bias and the complexity of the model. The assumptions required by autoregressive processes might not hold for the DHLAN. The method of replications, batch means and standardized time series can be applied to the DHLAN model. The applicability of these three techniques to the dynamic hierarchy network model motivates an examination/comparison of the three and the determination of the most suitable one for further experimentation. Half width of the confidence intervals for network

performance measures, coverage and cost in terms of central processing unit could be a few criteria for comparing the the three methods.

2.3 Simulation Models of LANs and Distributed Systems

An increasing demand for computer network services has widened the spectrum of networking and modeling and simulation. Simulation is an effective and a versatile method to examine complex problems. Subsystems of a computer network are simulated to analyze complex network problems such as, protocol layers, routing doctrines, flow control and congestion avoidance. Multiple modeling tools are now used in the various stages of computer network design, implementation, maintenance, and verification.

A simulation model for the dynamic hierarchy network model has been developed in Simula-67. The Simula process concept provides clarity in representing various forms of dynamic interaction among processes. It offers a *class* concept that facilitates system design. A suitable data structure and an action pattern can be defined for each class and an arbitrary number of instances of a class can exist at one time. Table 1.1 reviews prior simulation studies of LANs and distributed systems. Among the works reviewed in Simula, the study of the traffic handling capacity of Telephone networks and performance with respect to delay and throughput of simulated Data Communication networks show similarity to the study objectives of the DHLAN simulation. The objectives for simulating the Distributed Double Loop Network (DDLCN) in GPSS reflect much similarity to those of the simulated DHLAN model.

Table 1.1. Simulation Models of LANs and Distributed Systems

Ref	Title	Language	Objective
[1]	Simulation of SDL (Specification and Description language]	PASCAL	a) to study the common channel signaling system in the ISDN (Integrated Services Digital Network - 23 nodes, tree structure) by selection of suited data, b) study of different situations like the failure of a link, alternative routing, error detection to show the correctness of essential protocol functions.
[2]	Simulation Study of the Traffic Dependent performance of a prioritized, CSMA Broadcast Network.	ASPOL	a) to study the interactions of message size and configuration and its effect on network performance b) the effect on network performance of transfer rates through host interfaces of a CSMA network engaged exclusively in file transfer.
[3]	Modeling and Simulation of Data Communication Networks using SARA.	SARA (Systems Architects Apprentice)	to examine and explore the SARA system's usefulness in modeling and simulating a data communications network, where the network's performance is evaluated based on the queue length, queueing delay, storage utilization and processor utilization of each channel in the network.

Table 1.1. Simulation Models of LANs and Distributed Systems - continued

Ref	Title	Language	Objective
[4]	Simulation Study of the hyperchannel networks (i.e. hypernets).	ASPOL (process oriented language)	a) to study protocol interaction problems b) evaluate network's performance c) study trunk contention problems d) determine the overhead due to contention problems.
[5]	Simulation of a cable bus network in a multi-computer and large scale applications environment.	ECSS [Extendable Computer Simulation System] (superset of Simscript II.5)	to evaluate the impact of a particular cable bus technology on the performance of a large in service computer system, particular attention being paid to the magnitude and effects of network overload and the associated impact on computer capacity.
[6]	Real-time Simulation of a prioritized dynamic access protocol for LAN.	PASCAL	to develop a set of characteristics for the PDA protocol and its variants, the set of characteristics most likely being performance curves indicating the application environments into which the PDA protocol would best and worst fit.
[7]	Simulated Performance of a Ring-Switched Data Network.	GPSS	a) study and evaluate performance measures such as message delay, buffer occupancy and busy idle periods b) compare analytical results concerning message delay with results obtained from simulation c) investigation of message segments, line utilization and system capture by competing community of homogenous users.

Table 1.1. Simulation Models of LANs and Distributed Systems - continued

Ref	Title	Language	Objective
[8]	Simulation of the European Information Network (EIN).	SIMULA	a) to study the data transport in the EIN b) To design and implement the network switching centres. (note : the model can be reconfigured into an arbitrary network structure)
[9]	TETRASIM: A program study for the Simulation of telephone networks.	SIMULA	a) to investigate the properties of traffic flow in a large network b) to study alternative (eg. 8 node star shaped network) configurations, with increased number of inlet groups c) traffic handling capacity of the network.
[10]	Design and Simulation of the Distributed Loop Computer Network (DLCN)	GPSS/360	a) preliminary verification of the DLCN's performance predictions b) comparision of DLCN with Pierce and Newhall loops c) study of the queueing time and total message time.
[11]	Performance studies of fast Loop Networks.	CSIM (event driven simulation language in high level language C and Simula on PDP11/34)	to study the computer to computer transfer time for an improved type of token ring with acknowledgement support included in the network interface, for comparision the same analysis is done for an analogous token ring with acknowledgement handled solely by the host computers, performance is measured in terms of delay-throughput characteristics.

Table 1.1. Simulation Models of LANs and Distributed Systems - continued

Ref	Title	Language	Objective
[12]	A Simulated Data Communication Network.	SIMULA	a) to test a number of ideas concerning design decisions in planning a packet switching computer network b) to scrutinize the logical compatibility of the planned protocols of different levels, their parameterizing, and their performance with respect to throughput and delay c) test specific problems such as routing algorithms, protocols, flow control, trade-off between storage requirements at switching nodes and the channel capacity of links, topology and traffic rates.
[13]	A Homogenous Computer Network : Analysis and Simulation.	SIMSCRIPT II.5	a)to compare with analytical model for validation purposes b) to study a large number of problems varying transaction characteristics, remote traffic fraction, host configuration and communication subnetwork speed.
[14]	Analysis and Simulation of the Distributed Double Loop Network (DDLCN).	GPSS	a) to augment the results produced by the analytical model (link utilization and bottleneck analysis) with actual transmission time statistics b) compare DDLCN with DLCN c) investigate the effects of different message routing algorithms d) study the queueing and transmission times of messages.

Table 1.1. Simulation Models of LANs and Distributed Systems - continued

Ref	Title	Language	Objective
[15]	A generic program for the modeling and simulation of Distributed computer System Network.	SIMSCRIPT II.5	a) to enable the designer to select the distributed local network interconnections and run the conditions of his choice b) permit the individualized characterizations of each node c) permit simplified modifications to operational algorithms or the introduction of new ones.

The entries [1], [2], ..., [15] under the column Ref of Table 1.1 correspond to the following references:

[1] = [Hogrefe and Zorn 1986]

[2] = [Watson 1979]

[3] = [Razouk 1986]

[4] = [Leh 1979]

[5] = [Katkin and Sprung 1981]

[6] = [Dhadesugoor and Autenried 1986]

[7] = [Anderson 1972]

[8] = [Hansen 1978]

[9] = [Rogeberg 1978]

[10] = [Reames and Liu 1976]

[11] = [Dahlberg 1983]

[12] = [Gaspar and Lamm 1978]

[13] = [Labetoulle 1977]

[14] = [Wolf et al. 1979]

[15] = [Fenig and Myron 1979]

2.3.1 Summary

In summary, GPSS, Simula, Simscript, and Pascal are commonly used languages for network simulation. Most of the specifically designed simulation languages such as ECSS and CSIM have been derived from these languages. The objectives for most network simulations commonly relate to the investigation of some particular behavior of the network - a study of critical performance measures of the network verification of the network's performance predictions. For the DHLAN model, a study of the delay-throughput characteristics for several DHLAN networks and a comparison of the dynamically reconfigurable network architecture with its static counterpart is necessary. Clearly, the objectives for the DHLAN study are common to the objectives of other simulation studies of LANs and distributed systems. Performance measurement needs to be done by studying selected DHLAN networks under varying traffic demands and transition times. A comparative cost/benefits analysis can be accomplished by varying the traffic demands for both types, and varying the capacity assignment/apex only for the DHLAN.

3.0 DYNAMIC HIERARCHY NETWORK SIMULATION MODEL VERIFICATION

This chapter discusses the verification of the DHLAN simulation model and the experiment design to achieve the study objectives. The first section describes the salient features of the dynamic hierarchy network model as part of the network simulation model concept verification. The next section discusses the verification of the programmed model and the design of experiments.

3.1 Salient Features of the Dynamic Hierarchy Simulation Model

This section begins with a brief description of the dynamic hierarchy model structure followed by an overview of the important network components of the DHLAN simulation model. The key features of the model are supported with flowcharts as part of the network concept verification (communicative model verification).

3.1.1 Network Description

The dynamic hierarchy is an architectural variation of the centralized tree structured network. The hierarchical network consists of an apex, set of apex candidate nodes (ACS) and non-ACS nodes. At all times the apex is the logical head of the hierarchy. The ordinary nodes are connected to the ACS nodes in a strictly hierarchical topology. The nodes communicate with each other by sending messages. Following a store-and-forward organization messages are divided into packets of two types; control and data. Control messages have priority over the data messages. A predefined path joins all node pairs, and the packets are routed along

this established path. However, an event such as receipt of a control packet indicating the need for a transition might induce changes in the routing scheme.

The apex transition protocol defines the dynamic aspect of the hierarchical network. A transition results in the exchange of apex status between two ACS nodes. It also permits re-assignment of link capacities and variations in the set of active and inactive links. Active links have non-zero traffic rates while inactive links carry no traffic. The mechanics of network transition are described below.

3.1.2 Apex Transition Protocol

Three nodes play primary roles in the apex transition.

1. the initiating node (any network node) recognizes the external situation (change in traffic conditions) or an internal situation (partial or complete node failure) and decides to initiate a transition.
2. the incumbent apex arbitrates and manages the transition until the replacement apex takes over.
3. the new apex (from the ACS) following the transition assumes the logical head of the hierarchy.

Five control transport protocol data units (TPDUs) are exchanged during transition.

1. TR, the transition request sent by the initiating node to the incumbent apex node, contains the address of the new apex.
2. TV, transition vote, sent by the current apex to all apex candidates upon receipt of TR, serves to alert all the apex candidates of a transition in the future.
3. ACK, acknowledge, is sent by each apex candidate to inform the incumbent apex that a transition is accepted. The lack of acknowledgement represents an unaccepted decision. At this point, the apex can abort the transition.
4. TC, transition commit, is sent by the incumbent apex to all apex candidates to effect a transition. Each ACS node, upon receipt of the TC, interrupts data message transfer to

all ACS nodes and buffers the messages until the transition period ends. All data message traffic to the ordinary nodes continues, resembling a number of independent hierarchical LANs as illustrated by Figure 3.1. During transition, the routing tables for the ACS nodes are rearranged and new link transmission times are assigned to all the nodes.

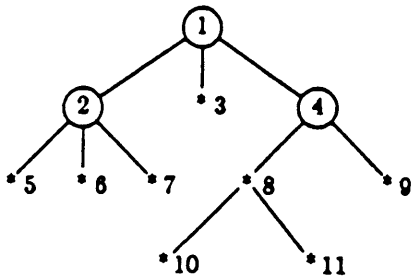
5. TO, transition over, sent by the apex to all the ACS nodes when the transition is over, at which time, the incumbent apex and the new apex switch roles. All previously buffered messages can now be transmitted [Bhat and Nance 1987; Moose and Nance 1987; Nagappan 1986; Moose 1983].

A few points of the transition protocol deserve to be noted:

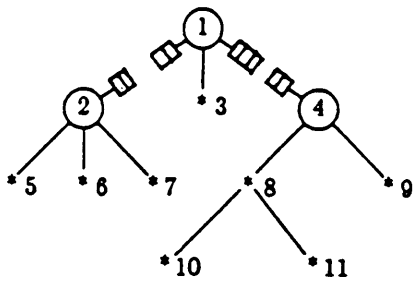
- Execution of a transition requires that all ACS members send a "yes" vote in response to the TV sent to them by the current apex. However, an ACS node can send a "no" vote thereby aborting the transition process. The present simulation model does not incorporate transition abortion, in other words an ACS node always votes "yes" for transition.
- All ordinary nodes are totally invisible to the transition.
- No visible changes occur in the configuration until the apex sends a transition commit to all ACS nodes, i.e. rearrangement of the routing tables, assignment of new link transmission times to all nodes and buffering of the data messages takes place only after the receipt of TC by the ACS nodes.

3.2 Overview of the DHLAN Simulation Model Structure

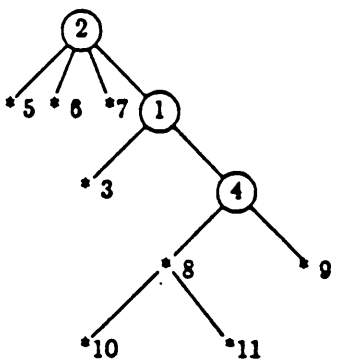
Figure 3.2 depicts the overall structure of the DHLAN simulation model. The important components of the simulation model are: packets, queues, nodes, configuration changer, statistics collector, network and, data analyzer. The key features of each component are de-



(a) *Regular Operation*: Node 1 as apex and nodes 2 and 4 being the other members of the apex candidate set (ACS). The need for reconfiguration is triggered by a "broadcast" from an ACS member.



(b) *Reconfiguration*: Each ACS node buffers all message packets to other ACS members but continues to serve its local hierarchy. Reconfiguration can be aborted by the incumbent apex until the final apex transition is signaled.



(c) *Adjustment*: Completion of reconfiguration initiates an adjustment period during which the messages buffered during the transition are transmitted. When all messages interrupted by reconfiguration are transmitted, a period of regular operation begins.

Figure 3.1. The Periods of Operation Induced by Reconfiguration: [MOOSE 87, p. 6]

scribed below. A more detailed description of the programmed model is given in Appendix C.

3.2.1 Packet

Nodes communicate with each other by sending and receiving packets. A packet contains the essential fields of:

- source node of the packet,
- destination node of the packet,
- type of the packet (Data, Request, Vote, Ack, Change, Over), and
- data (considered integer, and not a string of bytes).

In addition, some bookkeeping information such as:

- time the packet originated,
- sequence number of the packet, and
- pointer to the next packet in queue at each node.

3.2.2 Queue

This structure is used for all queueing mechanisms within the model. The queueing discipline is first in first out (FIFO). The following information is maintained for each queue:

- packets currently in the queue,
- total packets ever visited this queue,
- sum of the time spent by all packets ever visited the queue,
- last time the queue was modified by entry or exit of a packet,
- pointer to the first packet in the queue,
- pointer to the last packet in the queue, and
- maximum length this queue ever achieved.

The global operations allowed on the queue are:

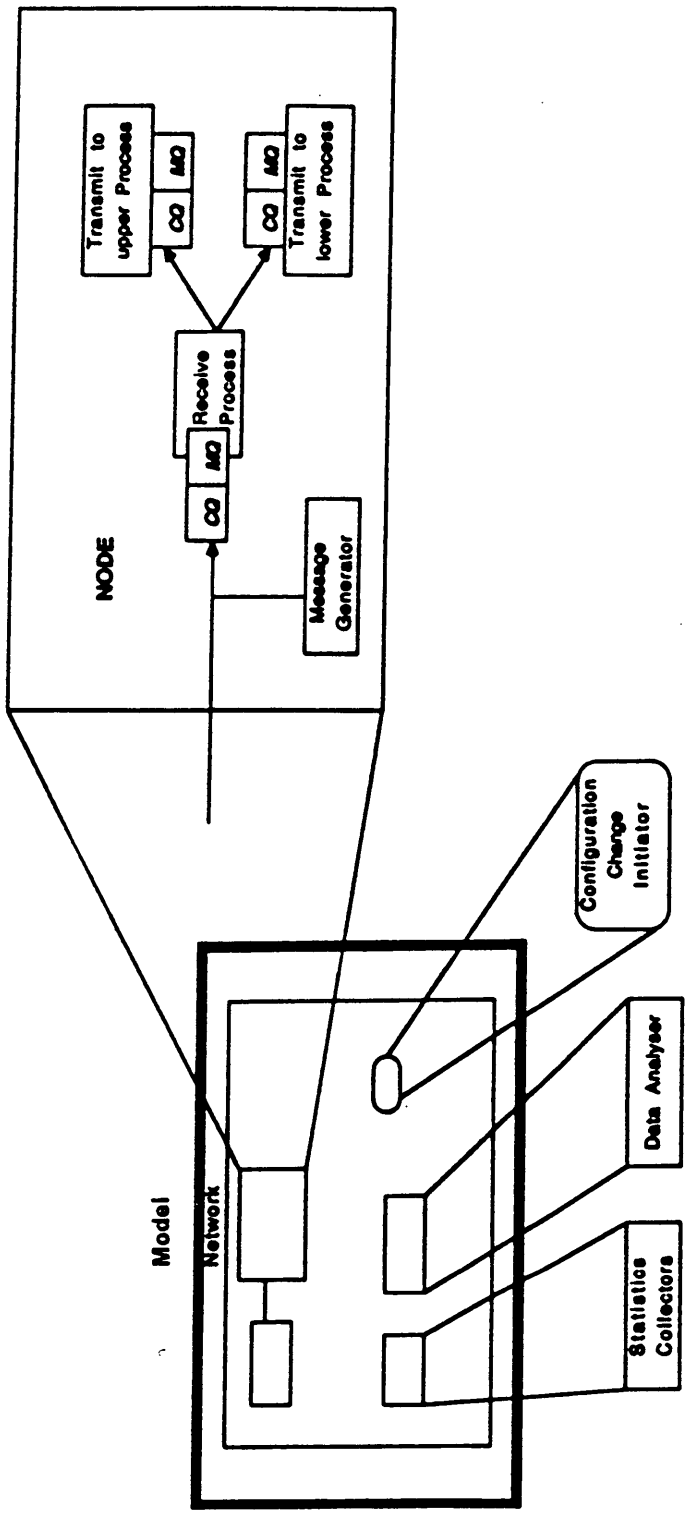


Figure 3.2. Structure of the DHLAN Simulation Model: Revised [NAGAPPAN 86, p. 31]

- appending a packet to the queue,
- removing the first packet from the queue, and
- testing if the queue is empty.

3.2.3 Node

The three types of nodes (apex, apex candidate, ordinary) differ only in their attributes. Each node consists of the following:

- three sets of queues, each set consisting of one control and one data queue. The first set corresponds to the input queues, second set to the transmit lower queues, and the third set to the transmit upper queues of the node. The queues are responsible for receiving messages from other nodes, and transmitting messages to upper and lower nodes. This capability is illustrated in Figure 3.3.
- one receive process, removes packets from the input queue and files them into the appropriate transmission queue. The receive processes differ for the three classes of nodes (apex, apex candidate, ordinary). Figures 3.4-3.6 illustrate the receive processes for the three node types.
- two transmit processes (upper and lower) responsible for removing the packets in a FCFS discipline from the transmission queues and routing them to the appropriate upper or lower control/data queue. The transmit process is shown in Figure 3.7.
- one message generator, producing outgoing messages and filing the packets in the input queue for the designated node as shown in Figure 3.8.
- address of predecessor node.
- routing table for communication with other nodes.
- number of packets originated at this node.
- inter-arrival time for messages generated at this node.
- link transmission time of the link connecting node and its predecessor.

- traffic matrix giving the average number of packets sent by this node to all other nodes in the network.

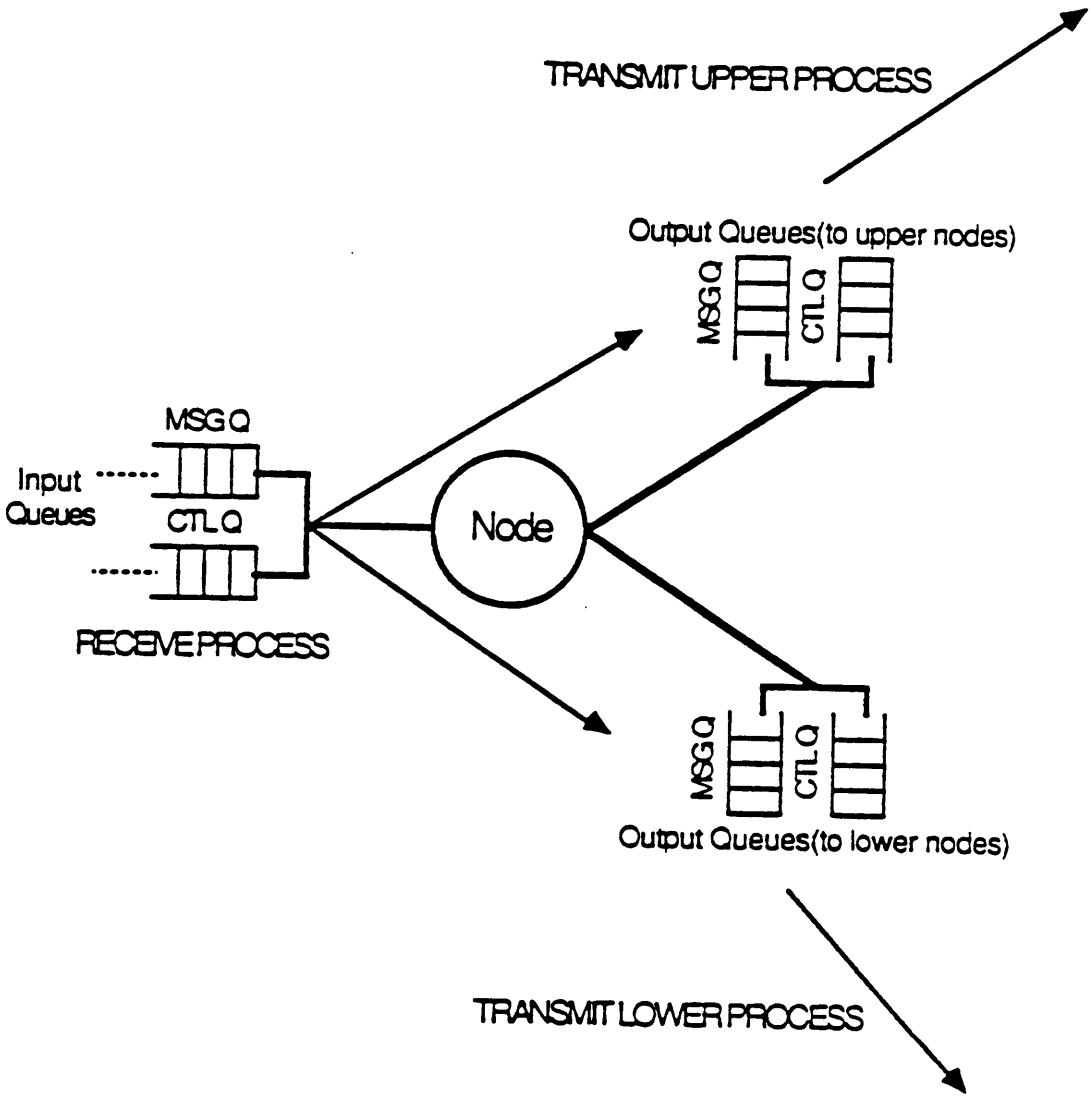


Figure 3.3. Queue and Process Structures: [NAGAPPAN 86, p. 26]

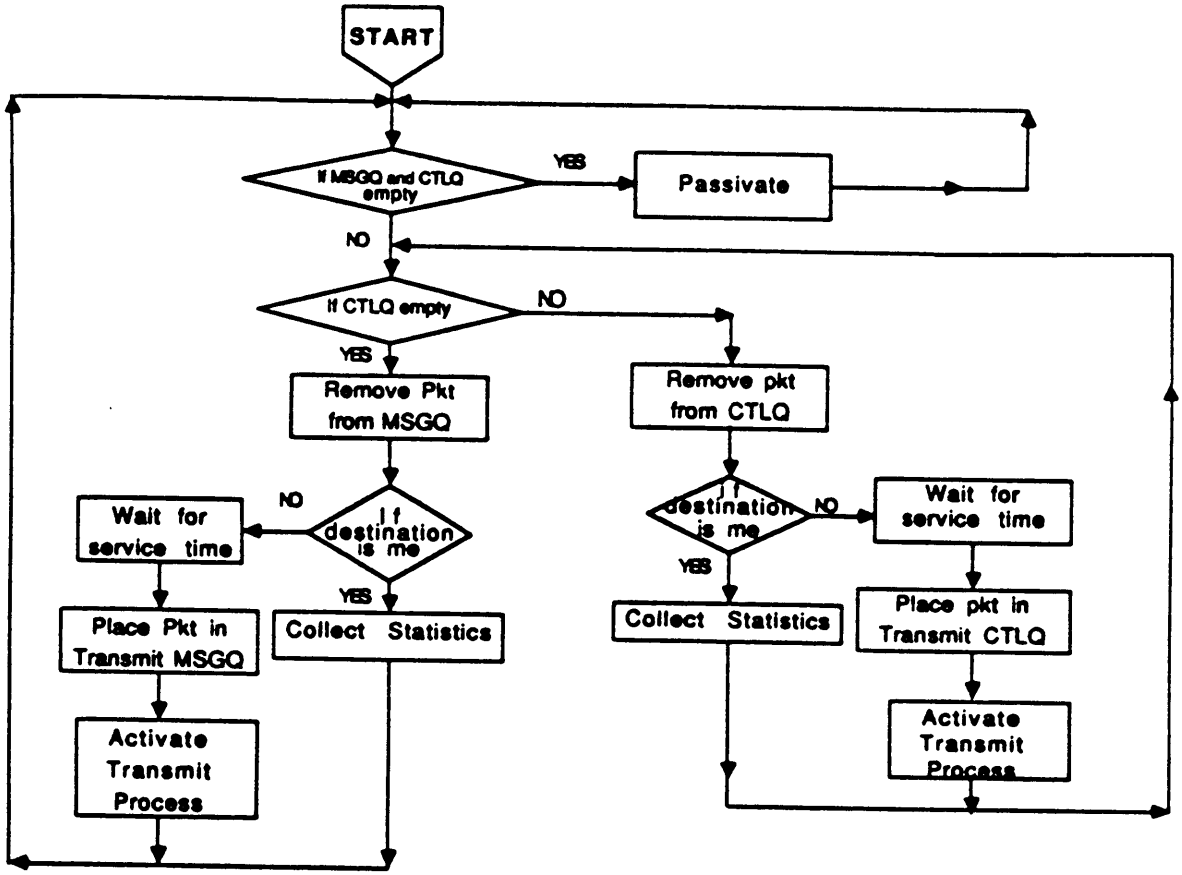


Figure 3.4. Receive Process for Ordinary Node: Revised [Nagappan 1986, p. 33]

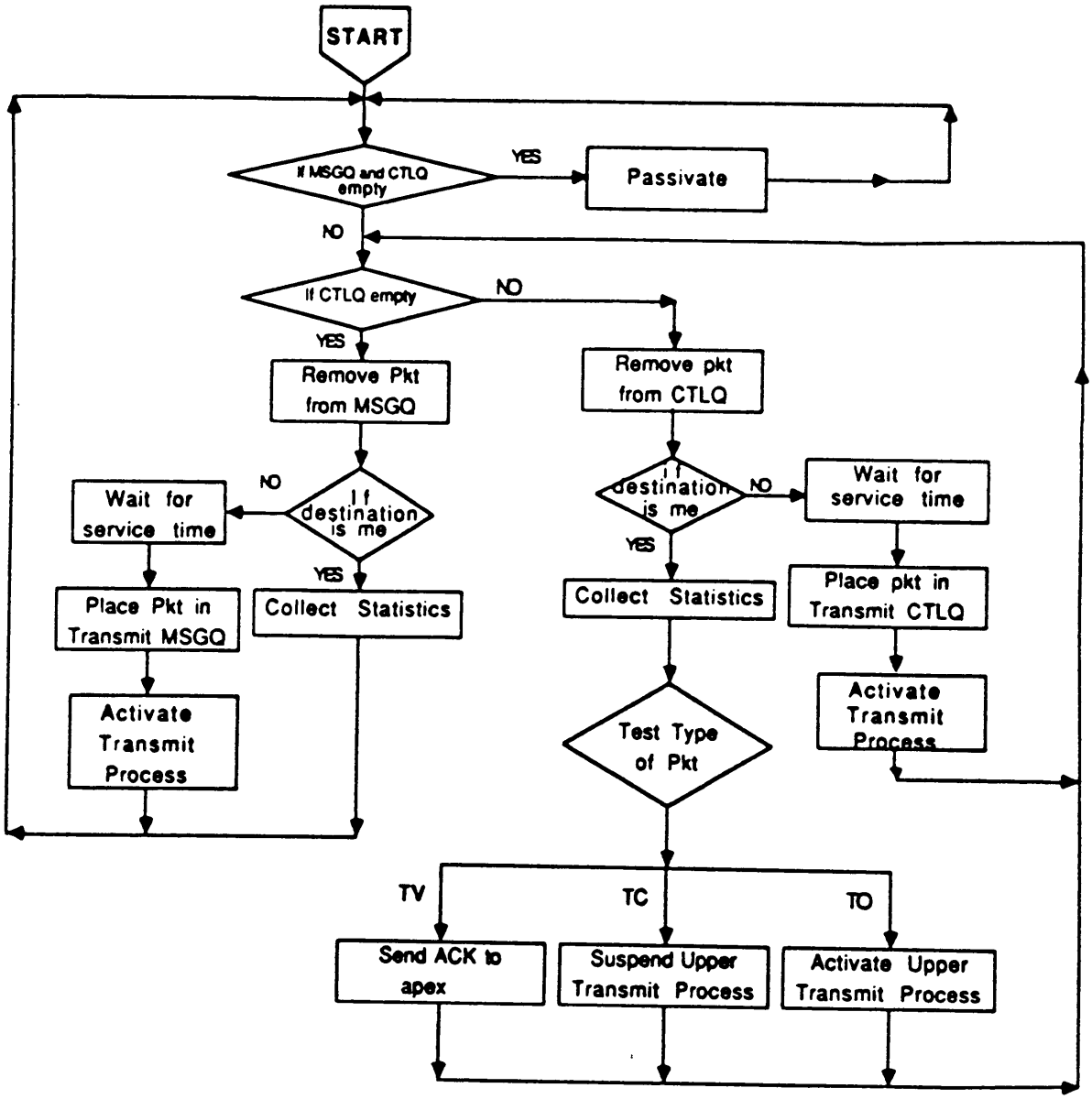


Figure 3.5. Receive Process for Apex Candidate Node: Revised [Nagappan 1986, p. 34]

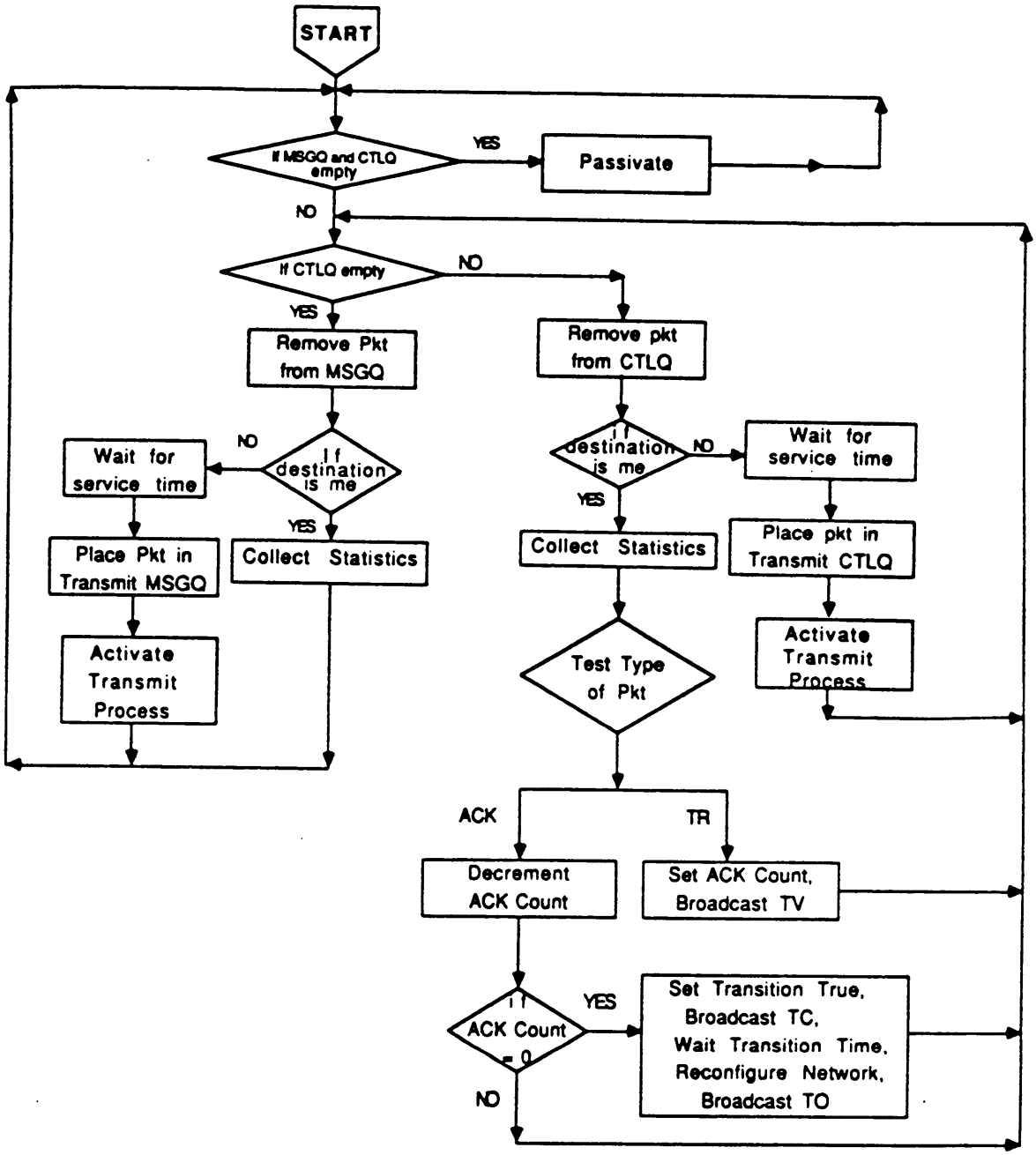


Figure 3.6. Receive Process for Apex Node: Revised [Nagappan 1986, p. 35]

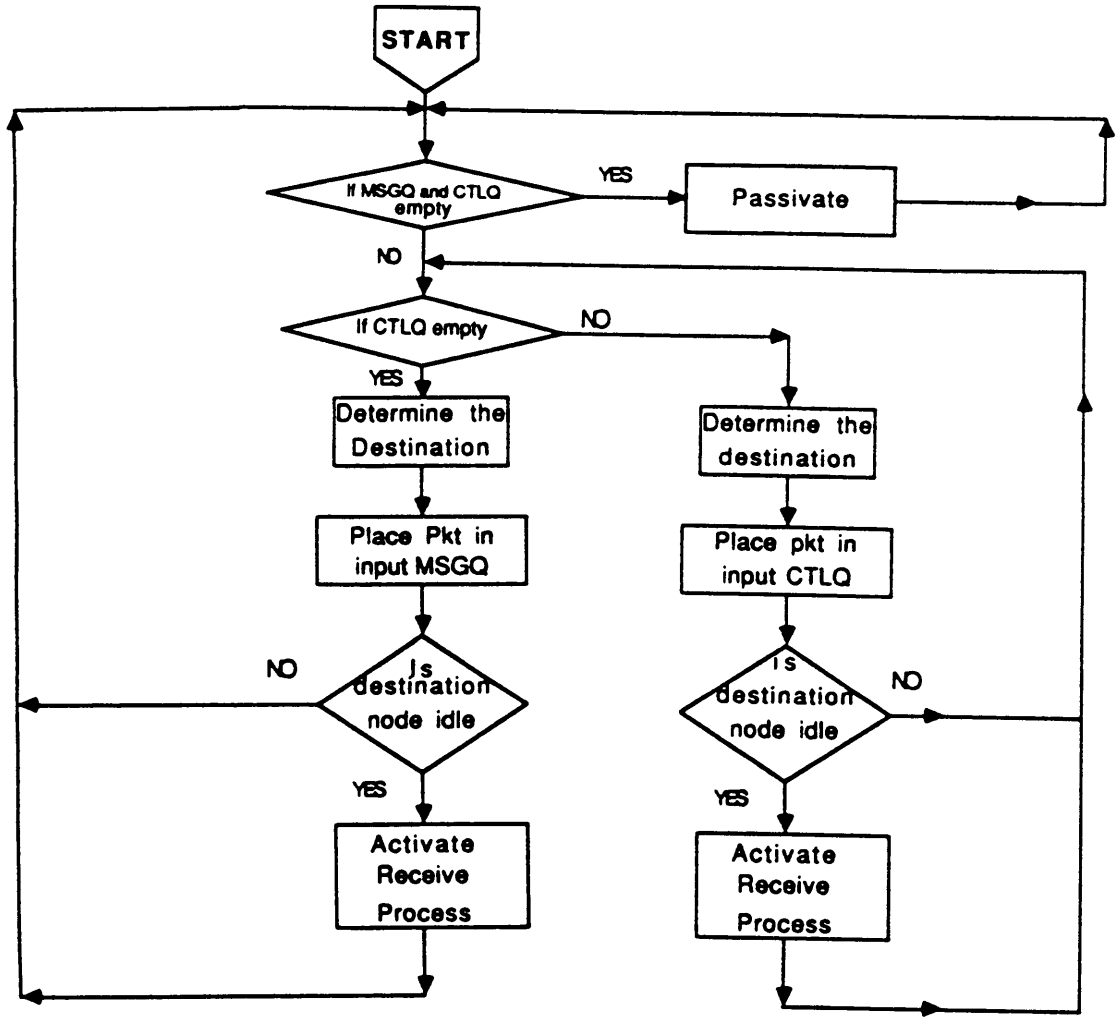


Figure 3.7. Transmit Process for a Node: Revised [Nagappan 1986, p. 36]

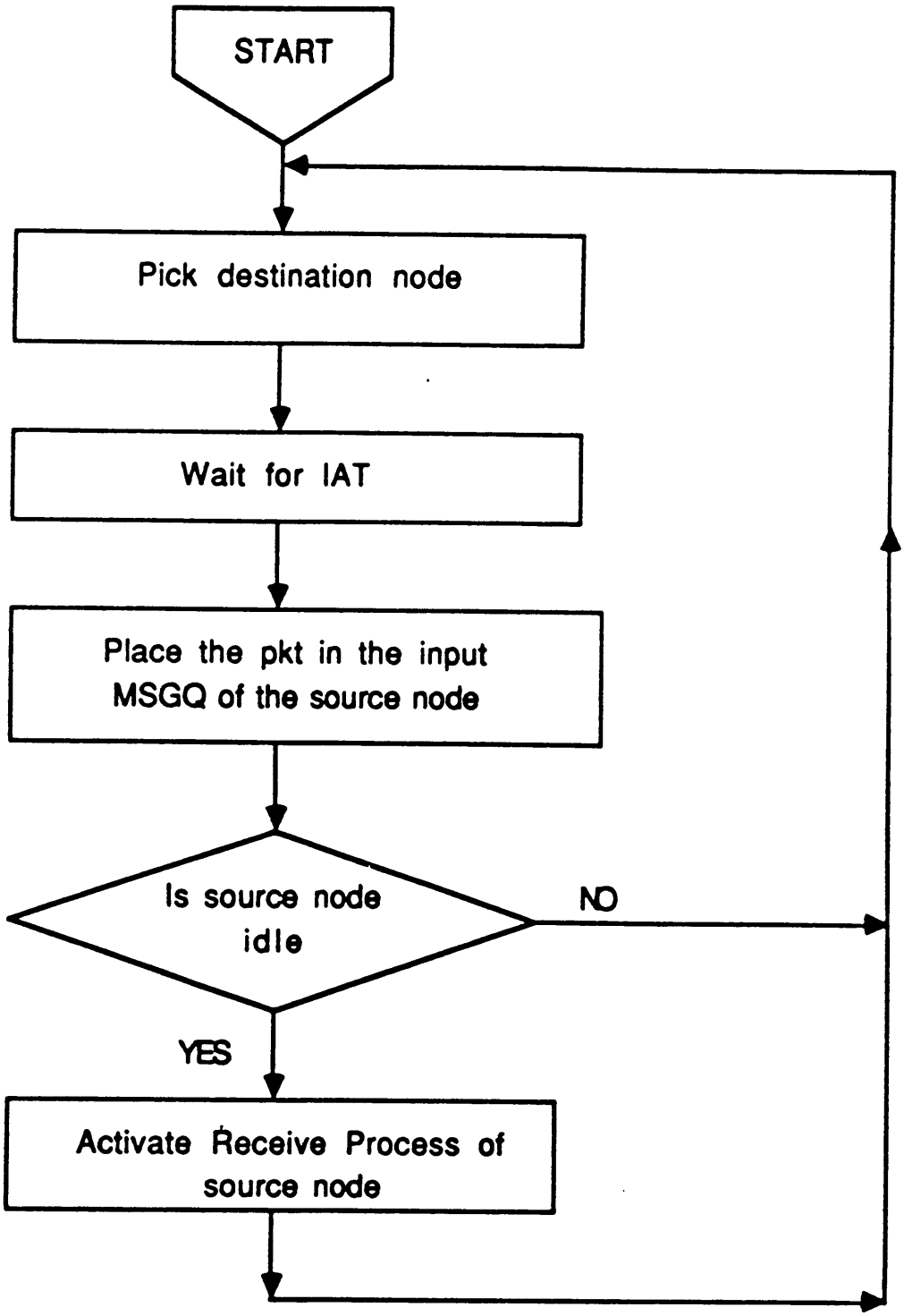


Figure 3.8. Message Generator for a Node: Revised [Nagappan 1986, p. 32]

3.2.4 *Configuration Changer*

The configuration changer is responsible for initiating a transition for reconfiguration of the hierarchy in response to external or internal conditions. The former is investigated exclusively in this work. The initiating node generates a control packet, stores transition request in its data field and sends the packet to the present apex signalling the need for a transition. This action is demonstrated in Figure 3.9.

3.2.5 *Statistics Collector*

Two components are responsible for collecting information on system behavior:

- The procedure rcd(packet) collects information on each packet delivered in the system. The point at which the stipulated number of packets have been delivered in the network for steady state conditions is signaled. When either the method of batch means or replications is used for output data analysis, it computes the link transmission delay for each delivered packet and keeps a record of the total delay incurred by all the packets delivered to that point. For the standardized time series method, in addition to the above, it stores the delay for each delivered packet (say packet i) and the cumulative average delay of the first i packets delivered in the system.
- The process daemon prints information on the sum of queue contents of ACS nodes at specified intervals of time. This provides useful insight into the queue behavior of ACS nodes during non-transition times and the queue overhead at these higher level nodes during the reconfiguration periods.

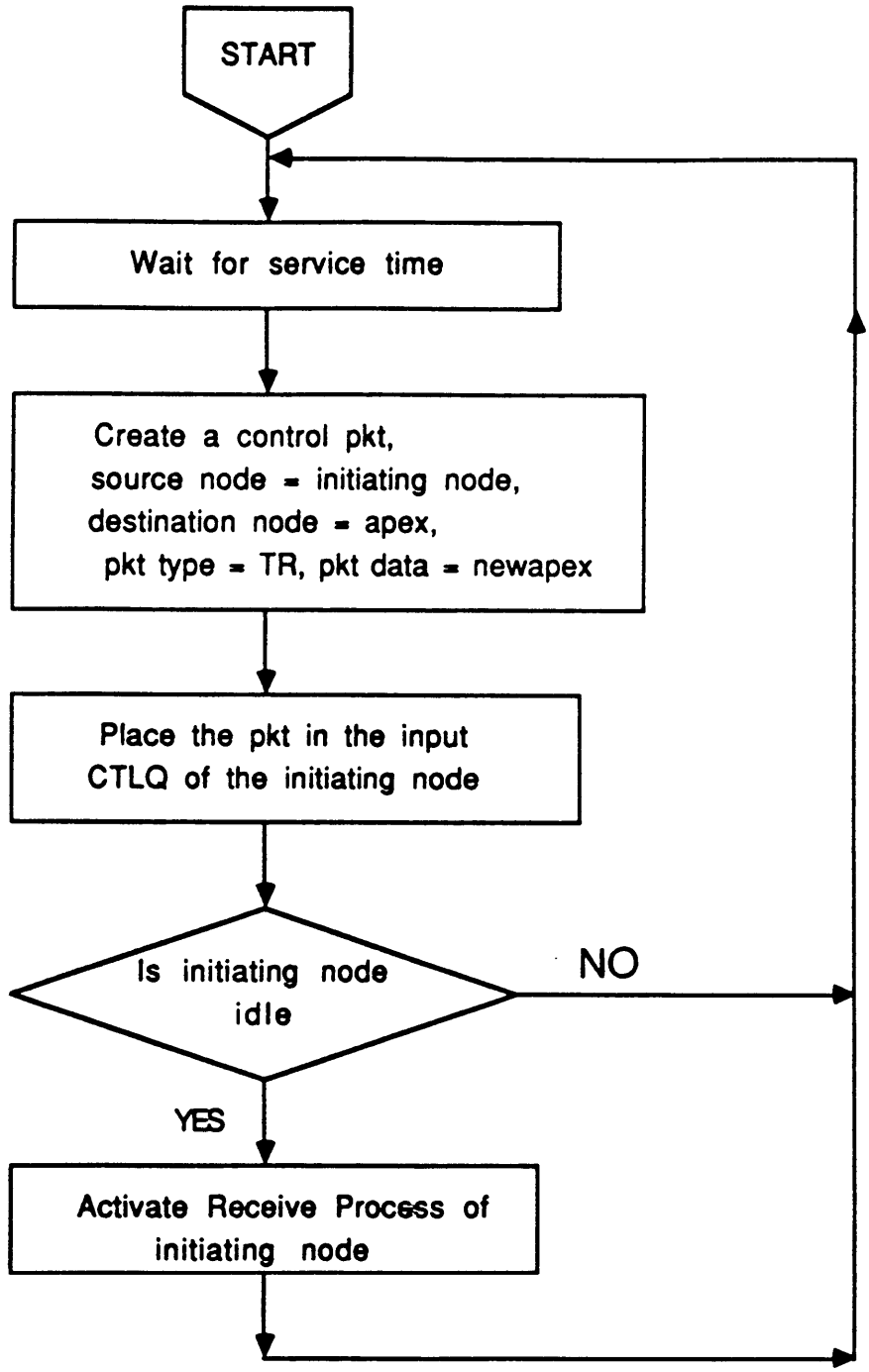


Figure 3.9. Configuration Changer: Revised [Nagappan 1986, p. 37]

3.2.6 *Network*

This is the core of the simulation model. Only one instance of this object exists at a time. It integrates all the network components and controls the interactions amongst the various processes. The network class operations include: nodes activations, establishing link connectivity, defining successor matrices and message routing tables.

3.2.7 *Data Analyzer*

The data analyzer processes the simulation output data according to one of three statistical methods: batch means, replications and standardized time series. The computations include the autocorrelation between the batch means and estimates of the mean, variance and confidence limits of the network parameters average delay, throughput and network delivery ratio.

3.3 DHLAN Programmed Model and Experiment Design Verification

The verification of the programmed model of the DHLAN and the design of experiments is described in the following paragraphs. Verification is accomplished using the techniques of desk checking, event occurrence trace checking and instrumentation-based testing.

3.3.1 *Verification Using Desk Checking*

Desk checking, a traditional form of software verification, consists of analyzing the model by hand [Glass 1979]. The DHLAN programmed model has been carefully analyzed by hand. Flowcharts of the network components have been thoroughly examined. The various network

components coded in Simula-67 are cross checked with corresponding system flowcharts and the accuracy of conversion has been verified.

3.3.2 The Trace Checking of Event Occurrences

An event is a change in object state occurring at an instant that initiates an activity precluded prior to that instant [Nance 1981, p. 31]. Verification of the correctness of the sequence of events in the DHLAN simulation model is accomplished by tracing and monitoring the model's dynamic behavior. Probes are embedded in the model to record the desired values. Event occurrences trace checking is an alternative form of instrumentation-based testing.

The DHLAN programmed model is instrumented, and execution of the model verifies the correct implementation of the following;

a. Each packet follows the correct message path from its originating node to its destination node.

b. When a packet reaches its destination node, statistics are accumulated properly and the packet is destroyed.

c. The queueing mechanism is correctly implemented and the packets are served according to the FCFS discipline.

d. All data packets at ACS nodes intended for higher level nodes are buffered during transition.

e. All data message traffic from ACS nodes to ordinary nodes continues during transition.

f. The ordinary nodes continue their normal operation during the transition period.

g. After transition, the ACS nodes stop buffering all higher level data traffic and switch to normal operation.

h. The desired ACS node assumes the role of apex at the end of the transition period.

3.3.3 *Instrumentation-Based Testing*

This technique requires insertion of probes into the programmed model to collect information about particular aspects of model behavior during execution [Probert 1982]. Several assumptions are made to verify the DHLAN programmed model and the experiment design.

a. Monitoring the network behavior starts upon the delivery of 50 packets in the network.

b. The daemon statistics accumulator and the configuration changer are activated when steady state conditions are achieved.

c. Transition time is 25.0 milliseconds.

A 6-node DHLAN is used for network verification. The model runs until it reaches the point at which network monitoring begins. At this point all accumulated statistics are deleted. Appendix A shows an example trace for the DHLAN programmed model verification and verification of the experiment design using the method of batch means for analyzing the simulation output data. The number of observations in each batch is equal to four hundred packets received at each of their final destinations. Traces utilized for the other two methods to verify the DHLAN programmed model and the design of experiments are not provided.

4.0 EXPERIMENTATION

4.1 Experiment Design

This section discusses the design of experiments to obtain required information to accomplish the following objectives:

1. comparison of the three statistical techniques for analyzing simulation results: batch means, replications, and standardized time series;
2. performance study of selected DHLANs; and
3. comparison of the dynamically reconfigurable network architecture with its static counterpart.

The issues of random number generation, normality, data independence and steady state analysis are addressed below. The formulation of the plans to determine the traffic matrices for the networks, capacity assignment to the links of a network and the packet arrival rate per node are discussed in the following subsections.

4.1.1 Random Number Generator

The random number generator standard for Simula-67 is used for all experimentation. The procedures for probability distribution use Lehmers multiplicative congruence method for random number generation:

$$U(i) = lambda \times U(i-1) \qquad (mod\ P)$$

$$X(i) = U(i)/P$$

with

$$P = 2^{32} = 4294957296$$

$$\text{lambda} = 5^{13} = 1220703125$$

Each X(i) is computed with 24 bits significance. Since the U(i) are represented with 32 value bits and no sign bit, the antithetic drawings are obtained by reversing the sign of the initial value [Hammarskjoldsvei 1986].

4.1.2 *Test for Normality*

The three output analysis methods (replications, batch means and standardized time series) assume the distribution of the sample means is normal. A departure from normality affects the confidence interval estimates for the steady state mean, μ. The Unifit package [Vincent and Law 1983] is used to test the normality of the sample means for the three methods for fitting probability distributions to the observed data. The standardized time series technique employs either the method of replications or batch means; hence, distribution fitting for replications and batch means suffices for the three.

Tables 4.1, 4.2, 4.3, and 4.4 give the results obtained for the tests performed using Unifit at the 95% confidence level. The entry "A" for "acceptable", corresponding to a distribution and the goodness-of-fit test, implies that the distribution seems to fit the observed data at the designated level of significance. An "M" (marginal) implies that the distribution seems to fit the observed data marginally at the observed level of significance. The designation "R" (reject) implies that the distribution does not fit the observed data. The probability distribution model with the smallest test statistic in most cases is chosen as the best fit of the distributions considered. The results clearly show that the normal distribution has the smallest test statistic under each of the three tests. Thus the model test comparisons heuristic suggests that the sample means are approximately normally distributed for $k \geq 10$, where k denotes the number of replicates or batches in a simulation run.

Table 4.1. Comparison of Test Results for Probabilistic Models: 10 Replications

Model No.	Probabilistic Model	Chi-square	Kolmogorov -Smirnov	Anderson -Darling
1	NORMAL	0.40000 A	0.18215 A	0.34687 A
2	EXPONENTIAL	1.60000 A	0.32127 A	0.80628 A
3	GAMMA	1.60000 A	0.31032 A	0.79049 A
4	LOGNORMAL	1.60000 R	0.32611 A	1.02787 A
5	WEIBULL	1.60000 A	0.29006 A	0.78571 A

Table 4.2. Comparison of Test Results for Probabilistic Models: 15 Replications

Model No.	Probabilistic Model	Chi-square	Kolmogorov -Smirnov	Anderson -Darling
1	NORMAL	0.40000 A	0.20003 A	0.59322 A
2	EXPONENTIAL	2.80000 R	0.34749 R	1.49217 A
3	GAMMA	1.20000 A	0.31947 A	1.28964 A
4	LOGNORMAL	0.40000 A	0.33865 R	1.64169 A
5	WEIBULL	0.40000 M	0.29003 A	1.20660 A

Table 4.3. Comparison of Test Results for Probabilistic Models: 10 Batches

Model No.	Probabilistic Model	Chi-square	Kolmogorov -Smirnov	Anderson -Darling
1	NORMAL	0.00000 A	0.14967 A	0.30701 A
2	EXPONENTIAL	0.40000 A	0.17617 A	0.60094 A
3	GAMMA	0.40000 A	0.18779 A	0.59257 A
4	LOGNORMAL	1.60000 A	0.25818 A	0.97433 A
5	WEIBULL	0.40000 A	0.17601 A	0.60030 A

Table 4.4. Comparison of Test Results for Probabilistic Models: 15 Batches

Model No.	Probabilistic Model	Chi-square	Kolmogorov -Smirnov	Anderson -Darling
1	NORMAL	2.80000 A	0.17098 A	0.47490 A
2	EXPONENTIAL	2.80000 M	0.23632 A	1.27695 A
3	GAMMA	2.80000 M	0.23360 A	1.24215 A
4	LOGNORMAL	3.60000 R	0.29639 M	1.90372 A
5	WEIBULL	2.80000 M	0.23052 A	1.11633 A

4.1.3 Test for Independence

A number of statistical tests used to analyze experimental data assume that the observations are independent and identically distributed. Independent replications can be achieved by using different initial seeds for each replicate. This is not true for the method of batch means. Batch independence is necessary to assure that the conditions required for a variance estimate are met. A small batch size has difficulty in satisfying the independence assumption. If a batch size is large, then the sample means are approximately normal and uncorrelated; hence the batch means may be treated as independent.

The batch size sufficient for passing independence for the experimental networks is determined using the method discussed below [Fishman 1978, pp. 237-240].

Let

m = number of observations/batch

k = number of batches in a simulation run

$\overline{Y}_{i,m}$ = sample mean for the i^{th} batch

$\overline{\overline{Y}}_k = \frac{1}{k}\sum_{i=1}^{k} \overline{Y}_{i,m}$ = estimator of the steady state mean μ

Consider the hypothesis:

H_0: $\overline{Y}_{1,m}, \overline{Y}_{2,m}, \ldots, \overline{Y}_{k,m}$ are i.i.d.

If $\{\overline{Y}_{i,m}\}$ has a monotonically decreasing autocovariance function $\{R_s \equiv cov(\overline{Y}_{i,m}, \overline{Y}_{i+s,m})\}$, then the null hypothesis H_0 is tested using a one sided test of at level of significance 2β. In particular, if

$$C_k \leq A_{2\beta} \qquad [4.1]$$

H_0 is accepted, otherwise it is rejected. Here

$$C_k = 1 - \frac{\sum_{i=1}^{k-1} (\bar{Y}_{i,m} - \bar{Y}_{i+1,m})^2}{2\sum_{i=1}^{k} (\bar{Y}_{i,m} - \bar{\bar{Y}}_k)^2}, \qquad [4.2]$$

$$A_{2\beta} = c(2\beta)\sqrt{(k-2)/(k^2-1)}, \text{ and} \qquad [4.3]$$

$$c(2\beta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{c(\alpha)} e^{-x^2/2} dx = 1 - \alpha/2 \qquad [4.4]$$

If $\{Y_{i,m}\}$ has a damped harmonic autocovariance function, i.e. the autocorrelation function assumes both positive and negative values, then a two sided test is performed. In this case, if $|C_k| \leq A_\beta$ then H_0 is not rejected. The estimates for the autocorrelation of lag j between the batch means is given by $\hat{\rho}_j$, where

$$\hat{\rho}_j = \frac{\sum_{i=1}^{k-j} [(\bar{Y}_{i,m} - \bar{\bar{Y}}_k)(\bar{Y}_{i+j,m} - \bar{\bar{Y}}_k)]}{\sum_{i=1}^{k} (\bar{Y}_{i,m} - \bar{\bar{Y}}_k)^2} \qquad [4.5]$$

A pilot run of an 11 node DHLAN simulation model shows the autocorrelation function of average delay assumes both positive and negative values as illustrated in Figure 4.1. Hence batch independence for DHLANs is determined using a two sided test [4.5]. Table 4.5 gives the results obtained for the 11 node network. This network is run for 15 batches, each batch having 2000 observations. Since $|C_k| \leq A_\beta$ for different batch runs, clearly a batch size of 2000 observations satisfies the independence hypothesis.

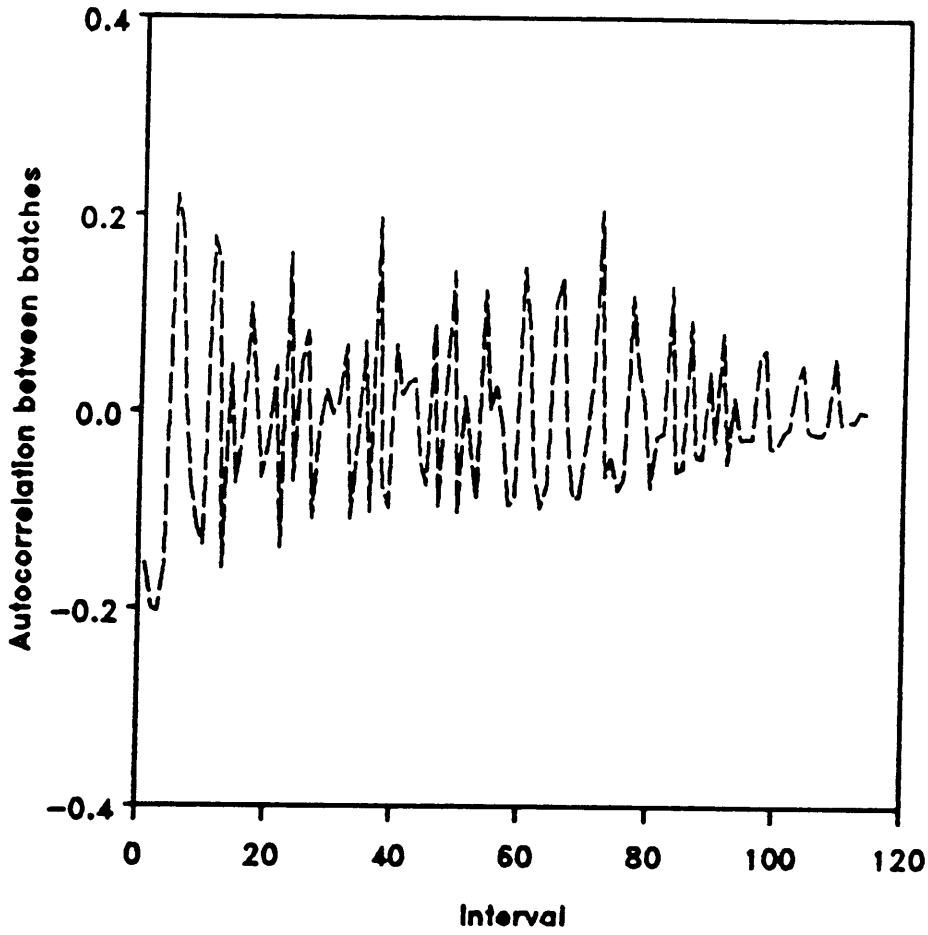


Figure 4.1. Autocorrelation Function of Average Delay: 11-Node DHLAN

Table 4.5. Test for Independence with batch size = 2000 observations

Number of Batches	$\|C_k\|$	A_β
10	0.26882	0.55716
20	0.01929	0.41629
30	0.03189	0.34590
40	0.00103	0.30215

4.1.4 *Steady State Analysis*

A system is said to reach steady state conditions, when the value of the system variables are independent of the initial conditions, and successive observations of the system's performance are statistically indistinguishable.

Queueing simulations are often started from an empty and idle state. An initial empty state might be correct for a terminating simulation; for example a bank simulated from opening until closing time. For a terminating simulation, the measures of system performance are defined relative to the interval of simulated time $[0,T_E]$, where T_E is the instant in the simulation when a specified event occurs [Law and Kelton 1982]. The terminating assumptions do not apply to the network model. The DHLAN model is to be analyzed for steady state behavior and the simulation run length is sufficient to obtain accurate estimates of the measures of interest.

The cumulative moving average method is used to determine steady state conditions for the DHLAN model. This method involves grouping the observations of a simulation into small intervals and using the group averages to compute a moving average of selected behavioral variables until they no longer change significantly over time [Emshoff and Sisson 1970]. DHLANs comprised of 5, 11, and 15 nodes are used for comparative analysis of network performance. Each network is run for 100,000 observations and batch means for the performance measures; average delay, throughput, and time based average delay/packet for ACS nodes are sampled at intervals of 1000 observations.

Several pilot runs are made for the three networks. Observations for the selected measures for each network are plotted in a cumulative moving average manner as shown in Figures 4.2-4.10. The vertical dotted line in each graph indicates the point at which this model in particular reaches steady state conditions for the stipulated performance measure. The parameterized version requiring the highest number of observations before achieving steady-state defines the beginning of measurement for all subsequent experimentation. Accordingly, this point is set at 45,000 packets and denoted by the solid line in each graph.

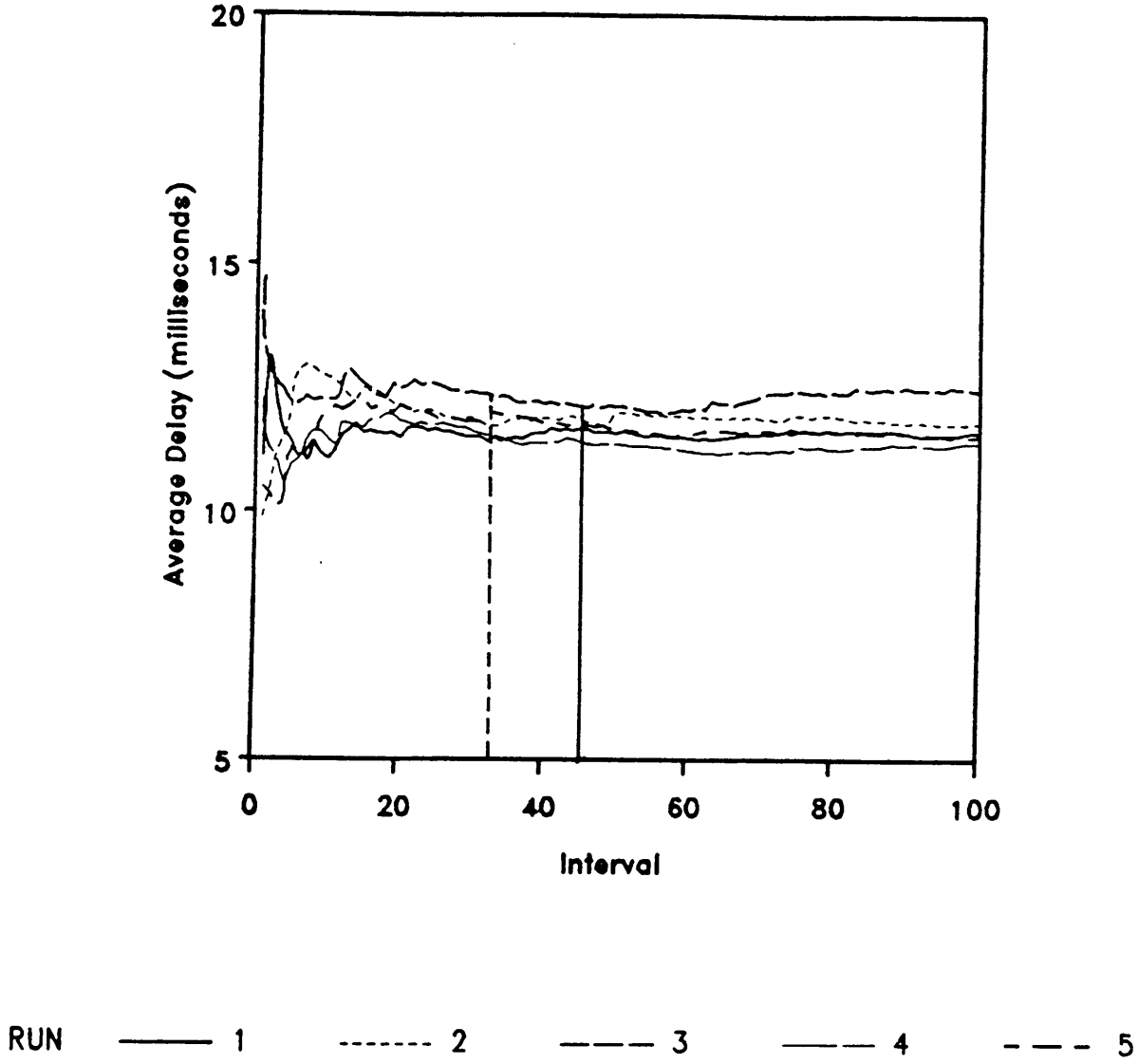


Figure 4.2. Cumulative Moving Average Estimate of Average Delay: 5-node DHLAN

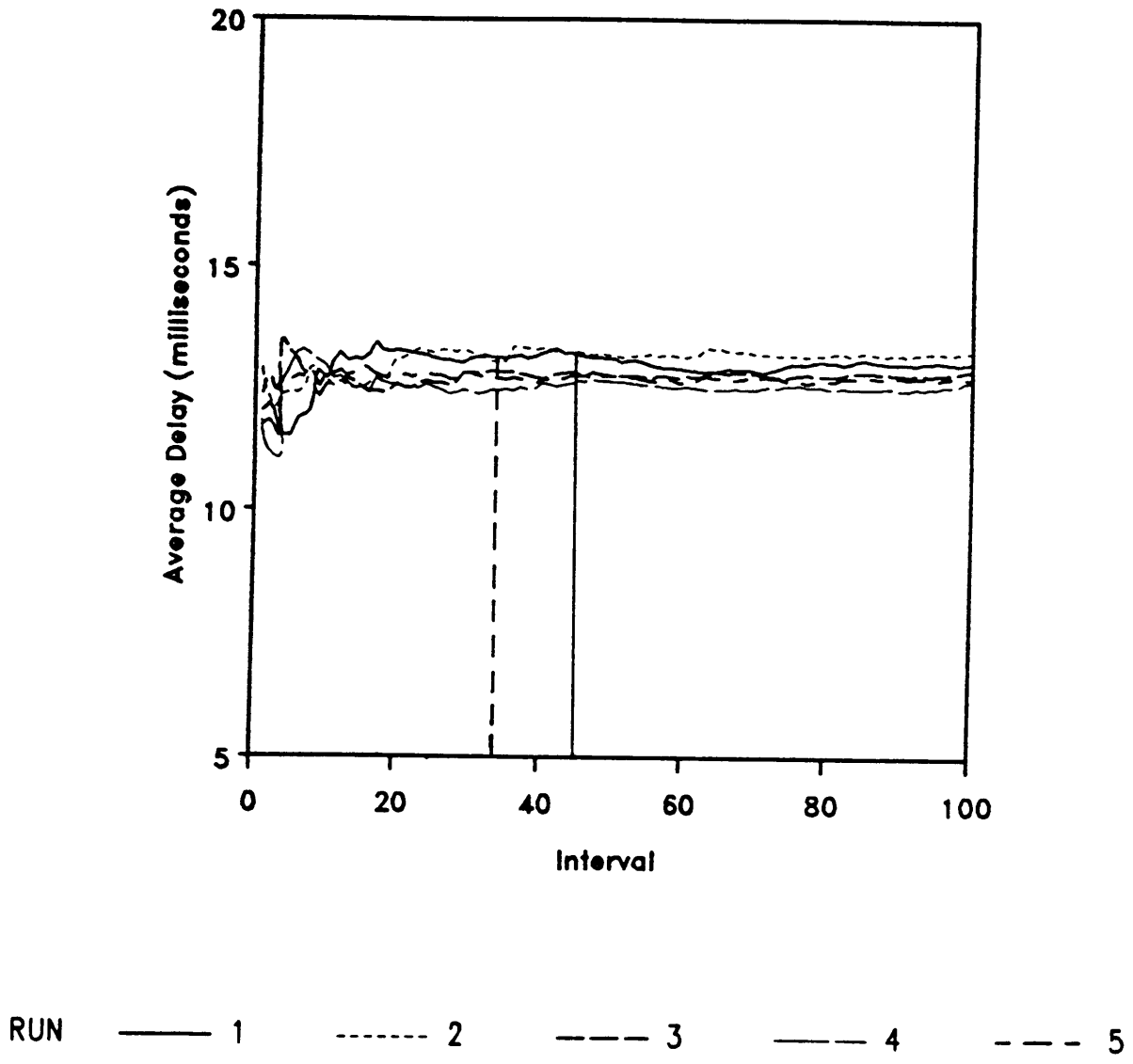


Figure 4.3. Cumulative Moving Average Estimate of Average Delay: 11-node DHLAN

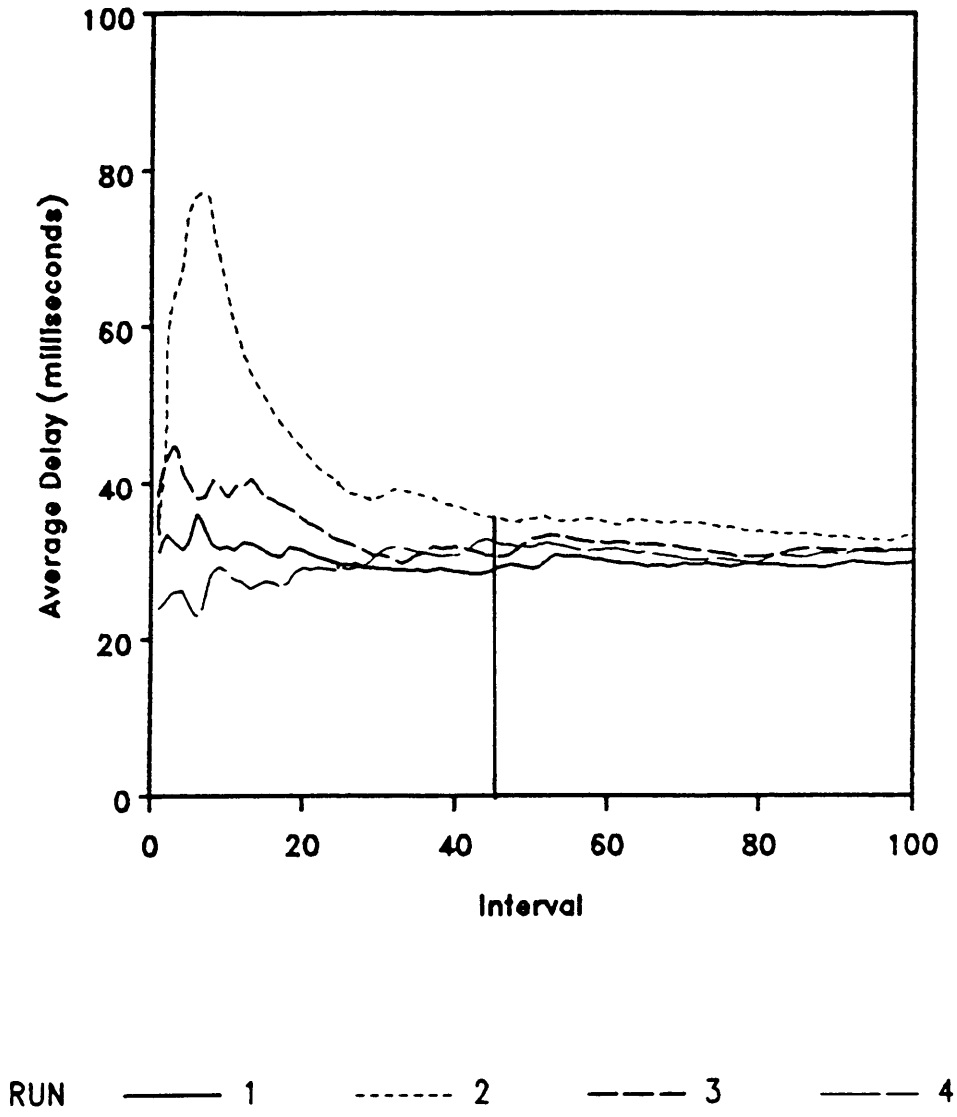


Figure 4.4. Cumulative Moving Average Estimate of Average Delay: 15-node DHLAN

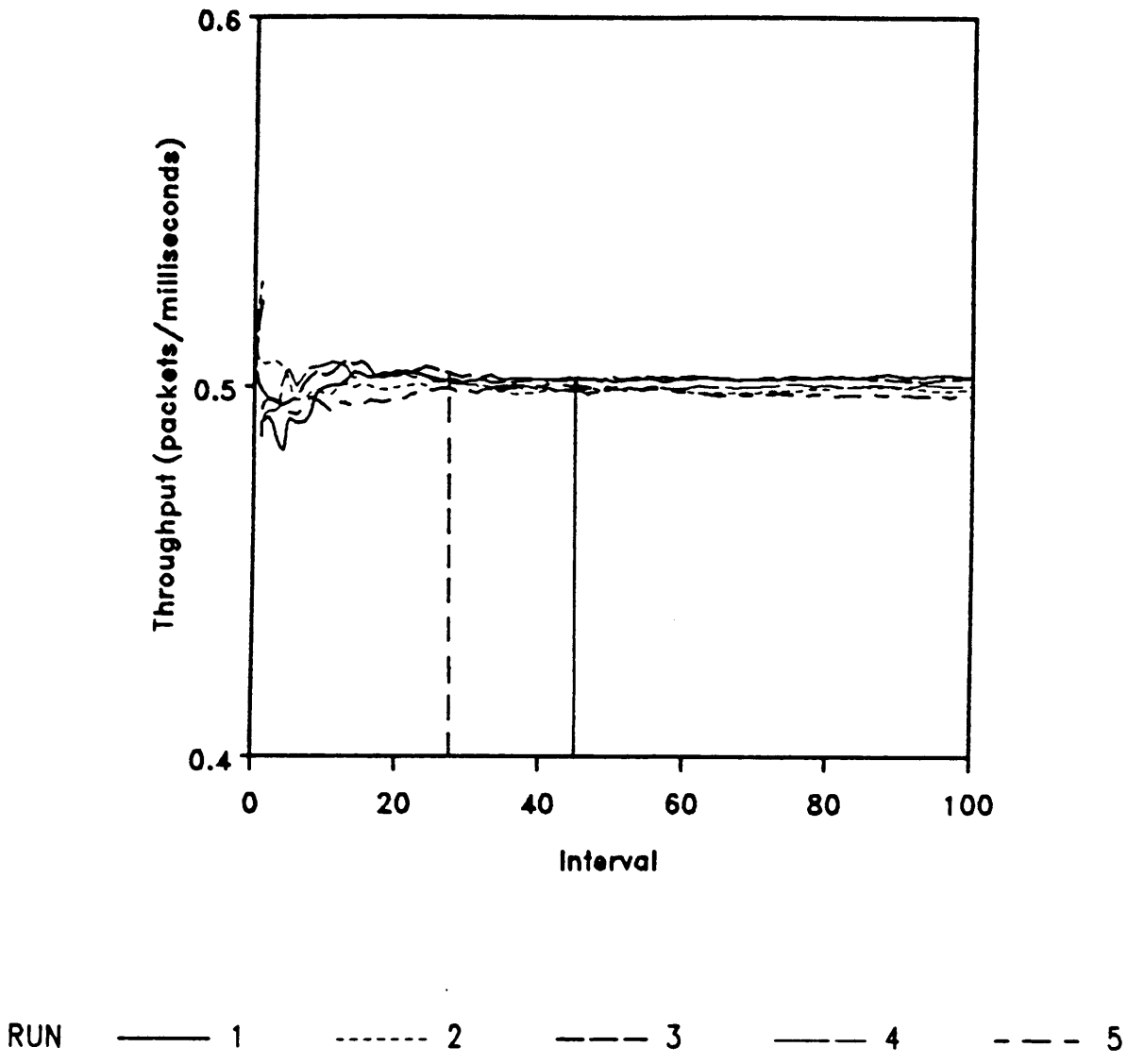


Figure 4.5. Cumulative Moving Average Estimate of Throughput: 5-node DHLAN

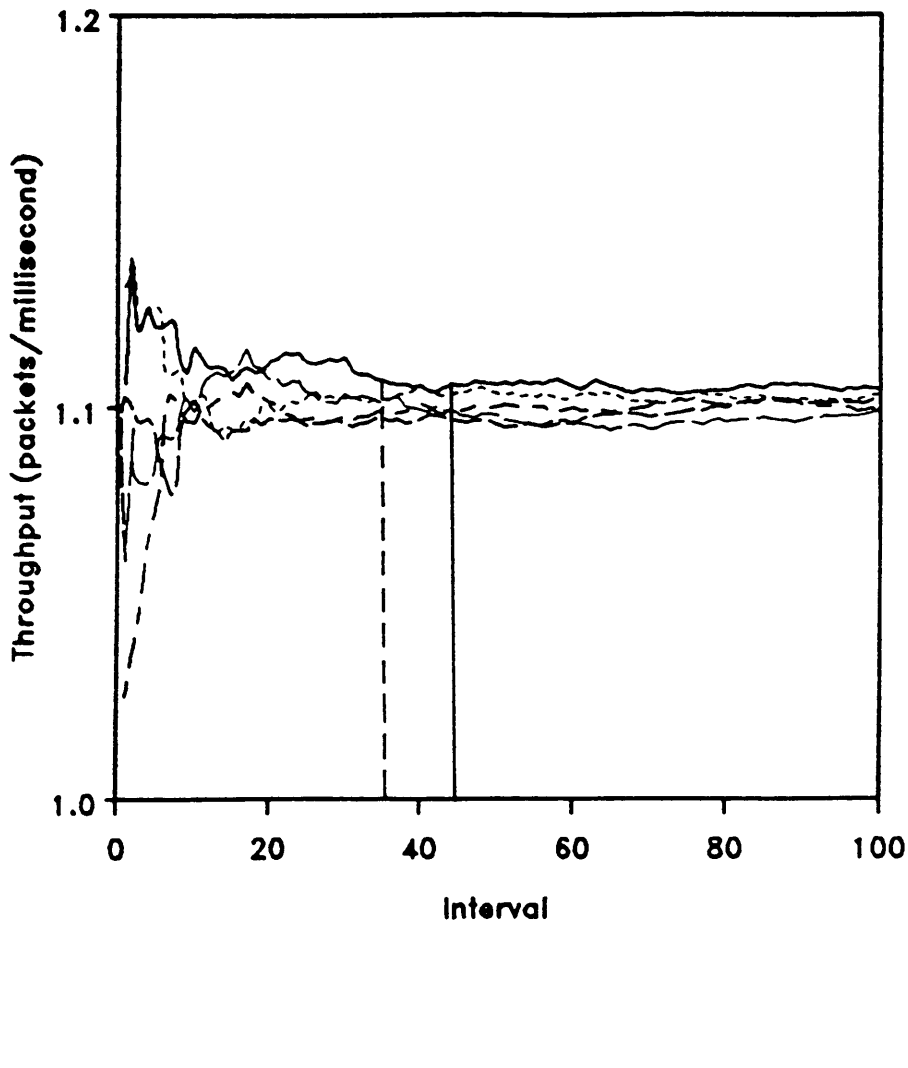


Figure 4.6. Cumulative Moving Average Estimate of Throughput: 11-node DHLAN

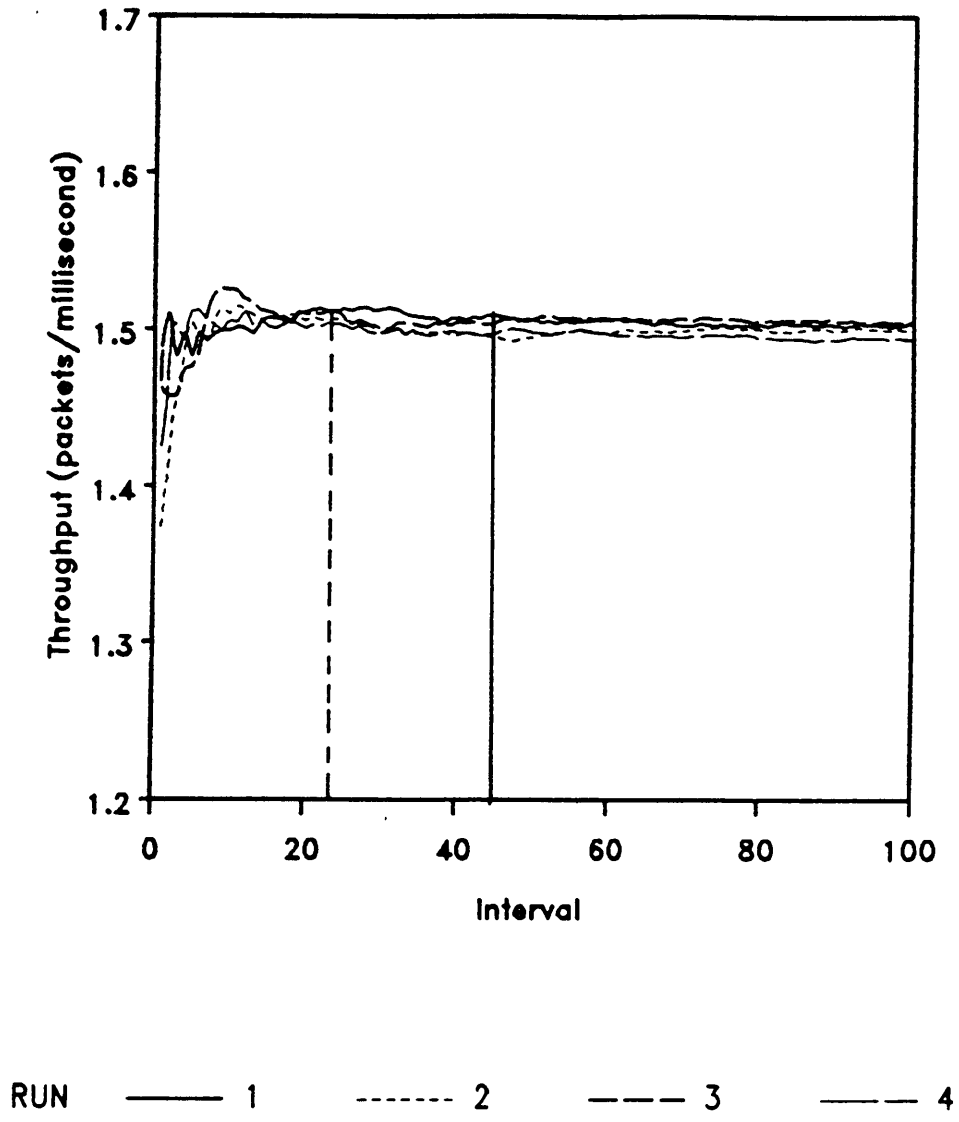


Figure 4.7. Cumulative Moving Average Estimate of Throughput: 15-node DHLAN

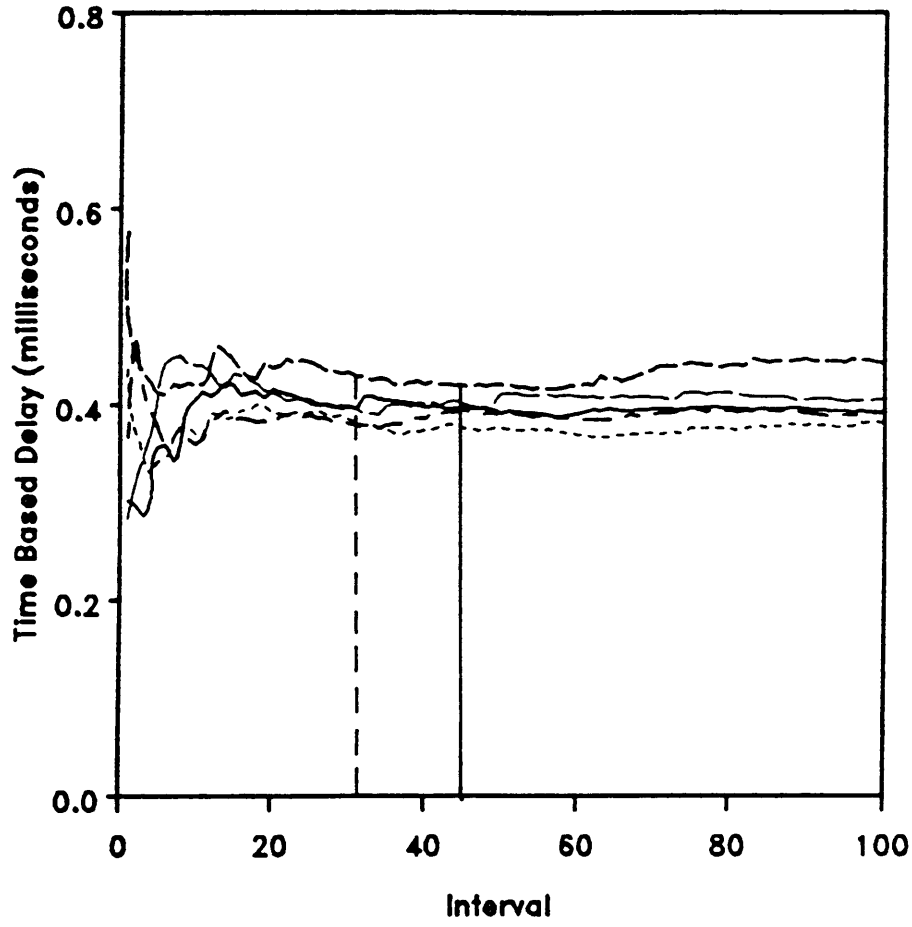


Figure 4.8. Cumulative Moving Average Estimate of Time Based Average Delay: 5-node DHLAN

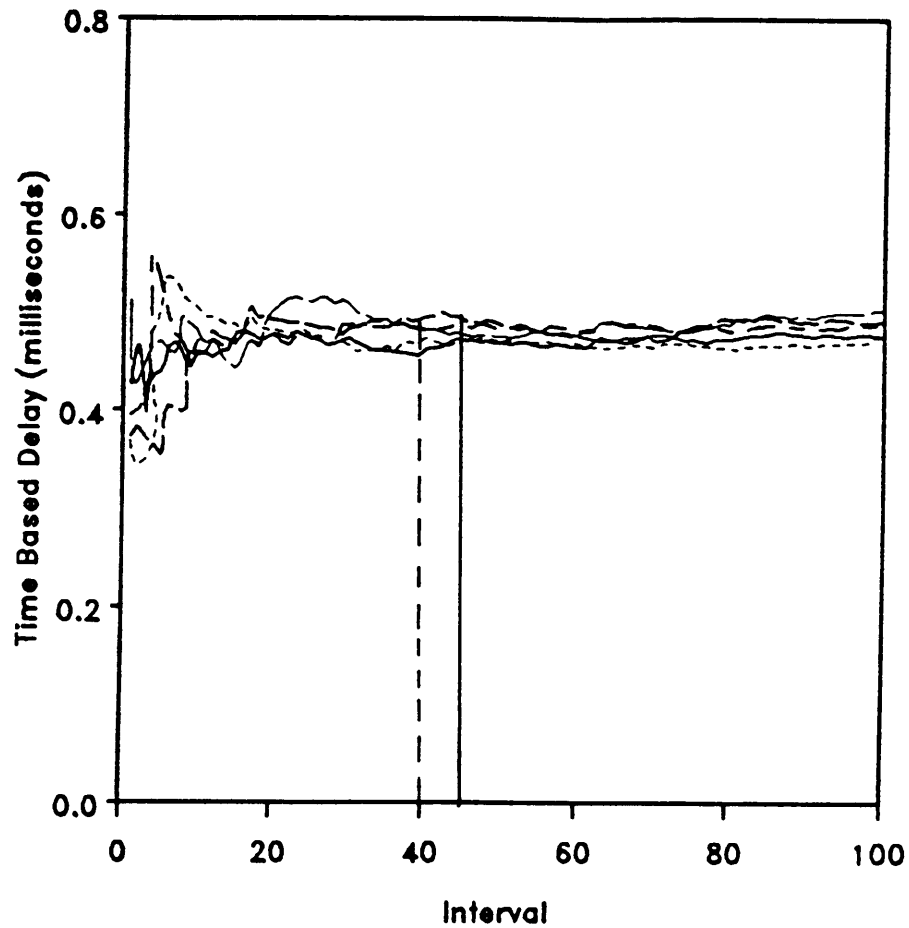


Figure 4.9. Cumulative Moving Average Estimate of Time Based Average Delay: 11-node DHLAN

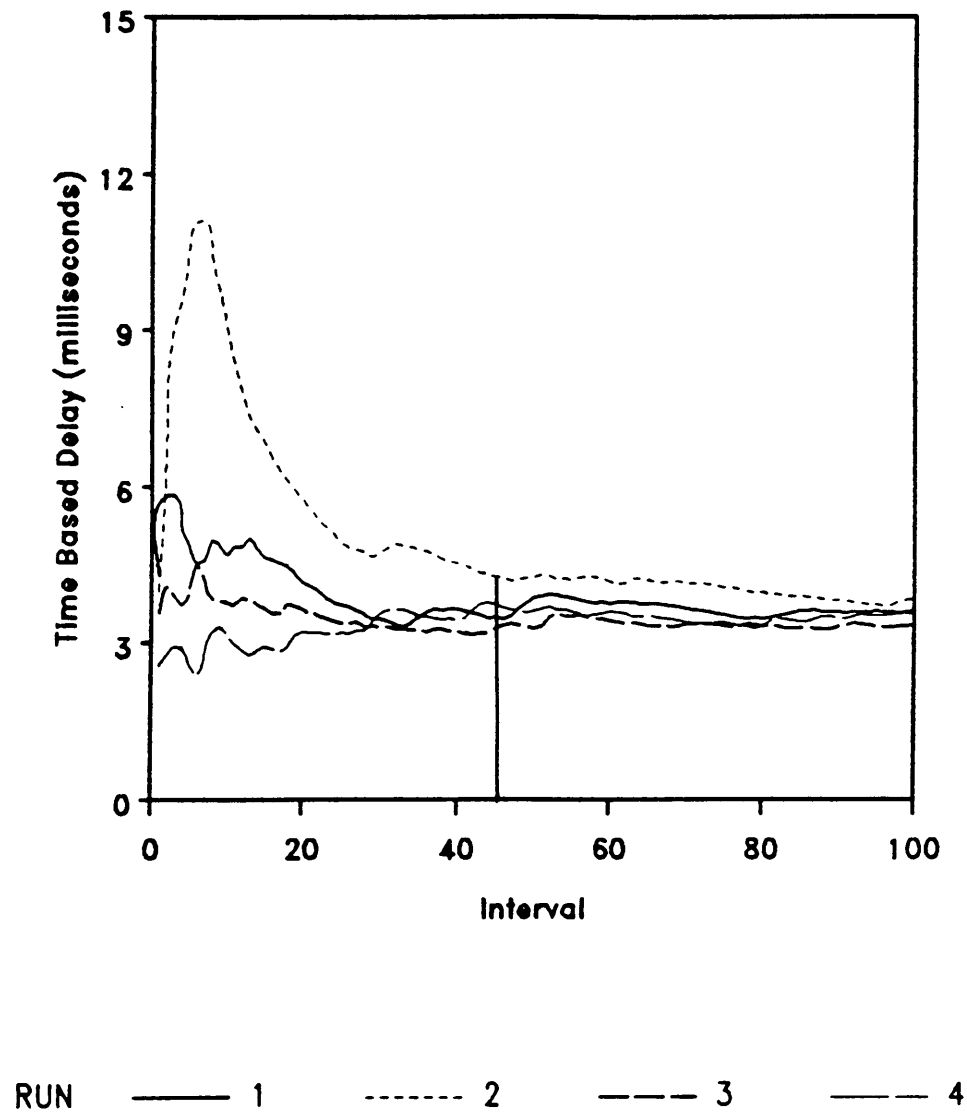


Figure 4.10. Cumulative Moving Average Estimate of Time Based Average Delay: 15-node DHLAN

4.1.5 *Traffic Matrix Determination*

Several assumptions are made in determining the traffic matrix for a network. Appendix B gives the traffic matrices that are used for experimental work. The assumptions are:

1. The total average message traffic emanating from any one node is 100 packets/second.
2. A node does not send any packet to itself.
3. Each traffic matrix represents one of several possible scenarios:
    A. A packet originating at one node is sent with equal equal probability to any other node in the network.
    B. The packet destination reflects some locality among subordinate nodes.
    C. The packet destination reflects heavy locality among subordinate nodes.
    D. A packet originating at one node is sent with a high probability to a node which is located across the apex node, i.e. more traffic routed through the apex.
    E. All nodes send 50% of their messages to an ACS node.

4.1.6 *Link Capacity Assignment*

Several assumptions are made in determining the link transmission rates for each link connecting the nodes [Schwartz 1977].

1. All links are full duplex and the link capacity in either direction is the same.
2. Traffic queueing for transmission over any link is statistically independent of traffic appearing anywhere else in the network, i.e. message lengths are independently generated at each node.
3. Channels are noiseless.
4. Nodes are reliable.
5. No priority discipline for serving the messages, (FCFS).

6. Messages have a single destination (no explicit broadcast capability).
7. Unlimited storage capacity at each node.
8. Packets are of variable length.
9. Nodal processing delay is negligible compared to the delay incurred waiting for the outgoing link to be made available.

4.1.6.1 Calculation of the Link Capacity Assignment

The capacity assignment strategy is illustrated with the help of an example. Consider a five node DHLAN as shown in Figure 4.11, and the following traffic matrix.

$$\begin{bmatrix} 0 & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & 0 & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & 0 & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & 0 & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & 0 \end{bmatrix} \qquad [4.6]$$

where a_{ij} = average rate of message flow between the node-pair (i,j). The average message rate λ_k destined for link k is the sum of all messages routed over that link. The values of λ_w, λ_x, λ_y, and λ_z are computed as follows:

$$\lambda_w = (a_{12} + a_{14} + a_{15}) + (a_{21} + a_{23}) + (a_{32} + a_{34} + a_{35}) + (a_{41} + a_{43}) + (a_{51} + a_{53}) \qquad [4.7]$$

$$\lambda_z = (a_{13}) + (a_{23}) + (a_{31} + a_{32} + a_{34} + a_{35}) + (a_{43}) + (a_{53}) \qquad [4.8]$$

$$\lambda_x = (a_{14}) + (a_{24}) + (a_{34}) + (a_{41} + a_{42} + a_{43} + a_{45}) + (a_{54}) \qquad [4.9]$$

$$\lambda_y = (a_{15}) + (a_{25}) + (a_{35}) + (a_{45}) + (a_{51} + a_{52} + a_{53} + a_{54}) \qquad [4.10]$$

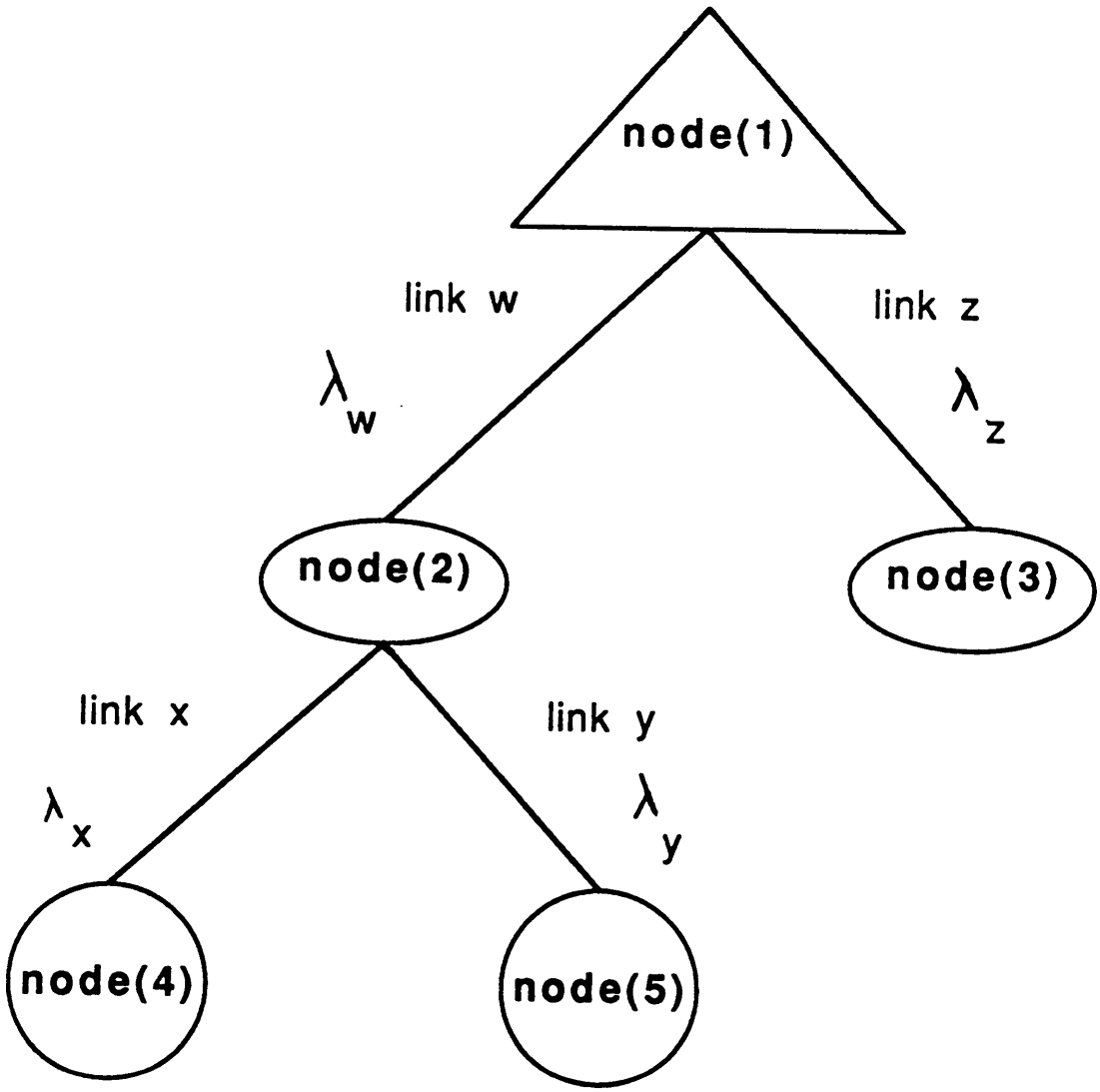


Figure 4.11. 5-Node DHLAN

The total average message traffic through each of the links for the example is given by $\lambda = \lambda_w + \lambda_x + \lambda_y + \lambda_z$ or in general if links are assigned numerical labels then $\lambda = \sum_{i=1}^{(n-1)} \lambda_i$, where n denotes the number of nodes in the network. A link's transmission delay (link service time) is represented by one of its nodes which is lower in the hierarchy. Since the apex node resides at the highest level in the hierarchy, the service time of the link connecting the apex with another node, say i, is represented by the node(2). Therefore the apex node is assigned a zero service time. In the network shown in Figure 4.11, node(2) simulates link w's transmission delay, node(3) simulates link z's transmission delay, node(4) simulates link x's transmission delay, and node(5) simulates link y's transmission delay. The link transmission delay for packets travelling to and from node(1) on link x are included in node(2). Similarly node(3) represents the delay for packets travelling to and from node(1) on link z.

Let $\rho_k = \lambda_k/\mu_k$ represent the utilization for link k , where μ_k = service rate at link k. For simplification purposes, for all networks, $\rho_k = 0.5$. Therefore $\mu_k = 2\lambda_k$, for $1 \leq k \leq (n-1)$, where n is the number of nodes in the network. The service times for the different scenarios are given in Appendix B.

4.1.7 Packet Arrival Rate At Each Node

A demand allocation scheme is used to assign packet arrival rates to each node in a network. To control congestion in the network, the total number of packets being generated is bounded. This bound is fixed according to the size of the network. Observing the bound, a higher packet arrival rate is assigned to designated heavy traffic nodes. Appendix B gives the set of packet arrival rates for the nodes comprising the experimental networks.

4.2 Design of Experiments to Compare the Three Methods

This section further discusses the experiment design to compare the three statistical techniques namely; batch means, replications, and standardized time series. The estimation of the parameter average delay and the confidence interval using the three techniques for the network parameters: average packet delay, throughput, and network delivery ratio are discussed in this section.

4.2.1 Definition of Network Performance Measures

The network performance is investigated with respect to the measures;

1. Average Delay

   Average delay is defined as the average link transmission delay of all packets delivered to their final destination. Link transmission delay of a packet is the time taken by a packet to reach its final destination from the time it originated.

2. Throughput

   Throughput is defined as the number of packets delivered to their final destination per millisecond in the network.

3. Network Delivery Ratio

   Network delivery ratio is defined as the ratio of the number of packets delivered to their final destination to the number of packets generated in the network.

4.2.2 Comparison of the Three Methods

A comparison of the methods of batch means, replications and standardized time series is based on the

1. half-width of the confidence intervals for the measure average delay
2. cost in terms of central processing unit (execution) time.
3. efficiency of the method in terms of its confidence interval containing μ and the confidence limits for μ, allowing 15% deviation for lower and upper confidence limits.

The absence of a real system, and the difficulty in building an equivalent analytical model, poses the problem of determining actual parameter values. The method of altered replications and variable sample extraction are two approaches chosen to estimate average delay and the confidence interval for a given level of significance. The method of altered replications relies on pilot run data to identify a state more representative of the steady state. This method has the advantages that a large number of replications gives a smaller variance estimate; hence a small confidence interval half-width, and a reduction in the transient phase length. The variable sample extraction method requires the simulation to run as one long batch and data be sampled at random intervals of time. This method has the advantage that as the run length increases, the stability of the model grows and therefore by randomly sampling over a very long period of time, a more precise estimate of μ and smaller estimates for the confidence interval of μ are expected.

The initial conditions for the method of altered replications are based on pilot runs of the model. After initialization, the model is "warmed" for 2,000 observations before the start of data collection. For the variable sample extraction method, the average delay is computed uniformly at intervals of (2000.0,8000.0) milliseconds. The mean of the sample observations is the estimate for average delay. Table 4.6 shows the results obtained for a pilot run of the DHLAN model using the above mentioned methods to estimate the parameter average delay. Clearly, the altered replications method performs better than the method of variable sample

extraction. For fewer observations, the variance estimate for altered replications is smaller than that for the variable sample extraction method by a factor of 21!. A smaller variance estimate implies a smaller half-width and confidence interval. Since the parameter values are unknown, a small confidence interval will provide greater precision for an effective comparison. Also a reduction in the transient length in each replication is considerably cost economical since the DHLAN has a large transient phase. Hence the method of altered replications is chosen as a basis for comparing the three techniques.

Table 4.6. Altered Replications versus Variable Sample Extraction

Method	No. of Observations	Mean	Variance	Confidence Interval	Half Width
Altered Replications	240,000	18.558	0.0588	(17.985,19.131)	0.573
Variable Sample Extraction	600,000	18.627	1.2587	(16.428,20.826)	2.199

A 11-node DHLAN is used for the comparative analysis. Using the method of altered replications, the 11-node DHLAN is simulated for 100 replications. Each replication consists of 45000 observations in steady state. Table 4.7 shows the results for this experiment.

Table 4.7. Estimation of Average Delay using Altered Replications: 11-Node DHLAN

Observations per Replication	Number of Replications	Average Delay	Variance	Half-Width	95% Confidence Interval
45000	100	19.191	0.00603	0.15225	(19.039,19.344)

4.2.3 Confidence Interval Estimation of the Network Parameters

This section discusses the procedures for confidence interval estimation for the methods of replications, batch means and standardized time series. For the methods of replications and batch means, once the sample means have been collected, the steps to estimate confidence intervals are identical.

4.2.3.1 Confidence Interval Estimation Using Method of Replications and Batch Means

Let

m = number of observations in a replicate or batch.

k = number of replications or batches.

n = total number of observations for the simulation run (n = mk)

Y(i,j) = j^{th} observation recorded in the i^{th} replicate/batch.

$\bar{Y}$ (i) = sample mean for the i^{th} replicate/batch, i.e.

$$\bar{Y}(i) = \frac{1}{m}\sum_{j=1}^{m} Y(i,j) \quad 1 \le i \le k \qquad [4.11]$$

$\bar{\bar{Y}}$ = point estimator of the steady state mean μ, is given by

$$\bar{\bar{Y}} = \frac{1}{k}\sum_{i=1}^{k} \bar{Y}(i) \qquad [4.12]$$

Estimate of the variance of $\bar{\bar{Y}}$ is given by

$$\hat{\sigma}^2(\bar{\bar{Y}}) = \frac{1}{k(k-1)}\sum_{i=1}^{k} [\bar{Y}(i) - \bar{\bar{Y}}]^2 \qquad [4.13]$$

An exact 100(1 - α)% confidence interval for μ is given by

$$\bar{\bar{Y}} \pm t_{k-1,\,1-\frac{\alpha}{2}} \sqrt{\hat{\sigma}^2[\bar{\bar{Y}}]}, \qquad [4.14]$$

where $t_{k-1,\,1-\frac{\alpha}{2}}$ is the 1 - α/2 point for a t distribution with k-1 degrees of freedom [Law 1980, pp. 668-669].

4.2.3.2 Estimation Using Standardized Time Series

Let

m = number of observations in a batch

k = number of batches

n = total number of observations for the simulation run (n = m.k)

Consider the i^{th} time series, $1 \le i \le k$, and let

Y(i,j) = j^{th} observation in the i^{th} batch.

$\bar{Y}$ (i,j) = cumulative average of the first j observations in the i^{th} series, i.e.

$$\bar{Y}(i,j) = \frac{1}{j}\sum_{p=1}^{j} Y(i,p) \quad 1 \le j \le m, \quad 1 \le i \le k \qquad [4.15]$$

$\bar{\bar{Y}}$ = sample mean for the entire data set, is given by

$$\bar{\bar{Y}} = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{m} Y(i,j) \qquad [4.16]$$

The time series is centered to have a mean of zero by transformation into the sequence S(i,j).

$$S(i,j) = \bar{Y}(i,m) - \bar{Y}(i,j) \quad 1 \le j \le m, \quad 1 \le i \le k \qquad [4.17]$$

For confidence interval estimation, the scaling constant cancels out, hence the magnitude of the sequence, S(i,j) is not scaled by dividing by sqrt(m)σ. The sequence S(i,j) is next converted to the standardized time series T(i,j).

$$T(i,j) = j \times S(i,j) \quad 1 \le j \le m, \quad 1 \le i \le k \qquad [4.18]$$

The scaled sum of the asymptotically normal random variables, T(i,j), is given by

$$\hat{A}(i) = \sum_{j=1}^{m} T(i,j) \quad 1 \le i \le k \qquad [4.19]$$

Define

$$\hat{A} = \sum_{i=1}^{b} \{12 \frac{\hat{A}^2(i)}{(m^3 - m)} + m(\bar{\bar{Y}}(n) - \bar{Y}(i,m))\} \qquad [4.20]$$

Interval estimation using the combined classical-sum interval estimator, I_{csum} for μ with confidence coefficient η is given by,

$$I_{csum} = [\bar{\bar{Y}} \pm H_{csum}] \qquad [4.21]$$

with half-width,

$$H_{csum} = t_{\eta,2k-1} \; sqrt\{\frac{\hat{A}}{n(2k-1)}\} \qquad [4.22]$$

where$t_{\eta,2k-1}$ is the 100(1 + η)/2 quantile of the t distribution with 2k-1 degrees of freedom. Interval estimation using the classical sum interval estimator, I_{sum} for μ with confidence coefficient η is given by,

$$I_{sum} = [\bar{\bar{Y}} \pm H_{sum}] \qquad [4.23]$$

with half-width,

$$H_{sum} = t_{\eta,k}\ sqrt\{12\sum_{i=1}^{k}\frac{\hat{A}^2(i)}{n^2(m^2-1)}\} \qquad [4.24]$$

where $t_{\eta,k}$ is the $100(1 + \eta)/2$ quantile of the t distribution with k degrees of freedom [Schruben 1983, pp. 1095-1100].

5.0 EXPERIMENTAL RESULTS AND DISCUSSION

This chapter discusses the results obtained through experimentation with the DHLAN simulation model. Section 1 discusses the experimental results for comparing the three statistical techniques: batch means, replications and standardized time series. Section 2 compares the cost and benefits of a dynamic hierarchy with its static counterpart. The third section discusses the performance of selected DHLANs with respect to the network parameters: average delay, throughput and network delivery ratio.

5.1 Comparison of the Three Methods

The 11-node DHLAN is used for comparative analysis of the three methods. This network is simulated for 40 replications/batches using each method at a time for output data analysis. Each batch or replication consisting of 45000 observations. The standardized time series relies on data collection using either the methods of batch means or replications, permitting estimation of confidence intervals in four different ways. The estimation of confidence intervals using standardized time series can employ either sample definition technique (batch means or replications). Data collected using batch means is analyzed using the classical sum interval estimator (denoted by STS(b-sum)) or the combined classical-sum confidence interval estimator (denoted by STS(b-csum)). STS(rep) denotes standardized time series using the method of replications.

Tables 5.1, 5.2, 5.3 and 5.4 show the experimental results. It should be noted that each experiment is performed as one long run and the desired estimates are printed at intervals of 5, 10, 15, 20 and 40 batches/replications.

Table 5.1. 95% Confidence Interval for Average Delay using Replications, Batch Means, Standardized Time Series for 11-Node DHLAN

k	Batch Means	Replications	STS (b-sum)	STS (rep)	STS (b-csum)
5	17.592,19.687	18.217,19.438	14.029,22.814	14.825,24.192	15.952,21.870
10	18.264,19.219	18.153,19.146	15.882,21.362	16.318,21.916	17.029,20.774
15	18.389,19.009	18.418,19.325	16.526,20.719	16.937,21.288	17.369,20.318
20	18.500,19.055	18.653,19.421	16.962,20.444	17.297,20.939	17.568,20.079
40	18.772,19.165	18.854,19.325	17.782,20.140	17.479,20.845	18.107,19.878

Table 5.2. Half-Width Estimates for Average Delay using Replications, Batch Means, Standardized Time Series for 11-Node DHLAN

k	Batch Means	Replications	STS (b-sum)	STS (rep)	STS (b-csum)
5	1.04753	0.61062	4.39281	4.68302	2.95939
10	0.47754	0.49636	2.74041	2.79890	1.87265
15	0.30997	0.45343	2.09632	2.17557	1.47446
20	0.27802	0.38392	1.74079	1.82106	1.25570
40	0.19620	0.23554	1.17912	1.68418	0.88546

Table 5.3. Mean Estimates for Average Delay using Replications, Batch Means, Standardized Time Series for 11-Node DHLAN

k	Batch Means	Replications	STS (b-sum)	STS (rep)	STS (b-csum)
5	18.63911	18.82721	18.42189	19.50900	18.91140
10	18.74178	18.64981	18.62202	19.11668	18.90170
15	18.69936	18.87158	18.62240	19.11253	18.84322
20	18.77664	19.03705	18.70279	19.11809	18.82399
40	18.96832	19.08936	18.96131	19.16356	18.99202

Table 5.4. Relative Cost in Terms of Processing Time in Minutes

k	Batch Means	Replications	STS (b-sum)	STS (rep)	STS (b-csum)
40	30	68	31	70	31

STS (rep) : denotes standardized time series using the method of replications for data collection.

STS (b-sum) : denotes standardized time series using the method of batch means for data collection and sum confidence interval estimator.

STS (b-csum) : denotes standardized time series using the method of batch means for data collection and combined classical-sum confidence interval estimator.

k : denotes the number of replications or batches.

5.1.1 *Discussion of Results*

The method of altered replications gives a small estimate of variance, hence small half-width and 95% confidence limits for μ, where μ denotes the parameter average delay (Table 4.7, section 4.2.2). Smaller estimates are desired to enable an effective and precise comparison of the three statistical techniques. The results for the different methods lead to the following observations and deductions.

1. For $k \geq 10$, batch means gives the smallest variance estimates of all the methods. This can be attributed to the tendency to reduce autocorrelation between neighboring batches as the separation between the observations increases.
2. STS(rep), STS(b-sum) and STS(b-csum) give larger estimates of variance and half-width for average delay. This is in direct contrast to the claim that standardized time series gives small estimates for half-width as compared to the classical methods of batch means and replications [Goldsman and Schruben, 1984]. During a simulation run, the DHLAN model undergoes complex process and object state transitions. The configuration change introduces a transient which is counter to the steady-state assumption. The standardized time series therefore produces larger interval half-widths in an attempt to preserve the prescribed coverage. This claim should be evident in the static case for STS. Since batch means and replications use only the means of each run to compute the estimates for the steady state mean μ, the transient effects have a reduced influence on the desired estimations. A comparative behavior of standardized time series and batch means is treated in more detail in section 5.1.4.
3. Estimates of variance and half-width for STS(b-sum) and STS(b-csum) are smaller than those for STS(rep). With batch means, the initial state for the run $(i+1)$ is the last state for the run i, eliminating initial condition bias as the run length increases. Hence, with each batch the model's stability increases resulting in reduced variabil-

ity in observations. In STS(rep) each run is independent of the other and the above phenomena is not observed.

4. Processing time is lower for all cases where data collection is done using batch means. Batch means eliminates initial bias except at the start of the single, long run; hence, a lower execution time for data collection results with this method.
5. STS(rep) gives the most consistent estimates for average delay.
6. All the methods with 5 batches/replications contain the confidence limits for μ which are obtained using the method of altered replications.
7. All variations of standardized time series contain μ and the confidence limits for μ for all values of k. This may be attributed to the large confidence interval estimates obtained with this method.

5.1.2 *Criteria for Comparing the Three Methods*

Several criteria are chosen for comparative analysis of the three methods. At this point, it should be kept in mind that small confidence interval estimates are desirable for prediction of the key network performance measure: average packet delay, termed average delay. The comparisons are based on

1. Confidence interval half-width for the measure average delay.
2. Coverage for average delay estimates established using altered replications.
3. Cost in terms of approximate processing time.

Five batches or replications are insufficient to satisfy approximate normality assumptions. Also, the half-widths corresponding to 5 runs is high for all methods rendering large confidence limits. Since batched/replication means are approximately normally distributed for $k \geq 10$ [section 4.1.2] the discussions are based on results obtained for $k \geq 10$ for the three methods.

5.1.2.1 Comparison Based on Half-Width

Clearly, the method of batch means gives the smallest confidence interval half-width for average delay. As earlier stated batch means performs better than replications as the effects of autocorrelation between adjacent batches diminishes as the run length increases. The weakness of STS to stationarity tends to give higher half-widths than those for batch means and replications.

5.1.2.2 Comparison Based on Coverage

Coverage is defined as the proportion of the confidence intervals that contain the confidence limits for average delay established using method of altered replications. A less stringent definition for coverage is the proportion of the confidence intervals that contain the average delay established using method of altered replications. Clearly all confidence intervals for all variations of STS contain μ and the confidence limits for μ exhibiting 100% coverage. Figure 5.1 shows the confidence interval plots for all the methods.

Since the STS gives wide confidence intervals which is not desirable, the coverages obtained from batch means and replications is studied. All confidence intervals for both the methods contain the lower confidence limits for μ. The method of replications gives one out of four confidence intervals that contain the confidence limits for μ and three out of four (75%) confidence intervals that contain μ. The method of batch means gives one out of four confidence intervals that contain μ and none that contain the confidence limits for μ. Therefore replications performs best for this criterion.

5.1.2.3 Comparison Based on Processing Time

Clearly methods of batch means and standardized time series (STS) using batch means for data collection consume the lowest processing time. Although batch means uses lesser time than STS, the difference is not significant. Therefore these two methods are equally desirable according to this criterion.

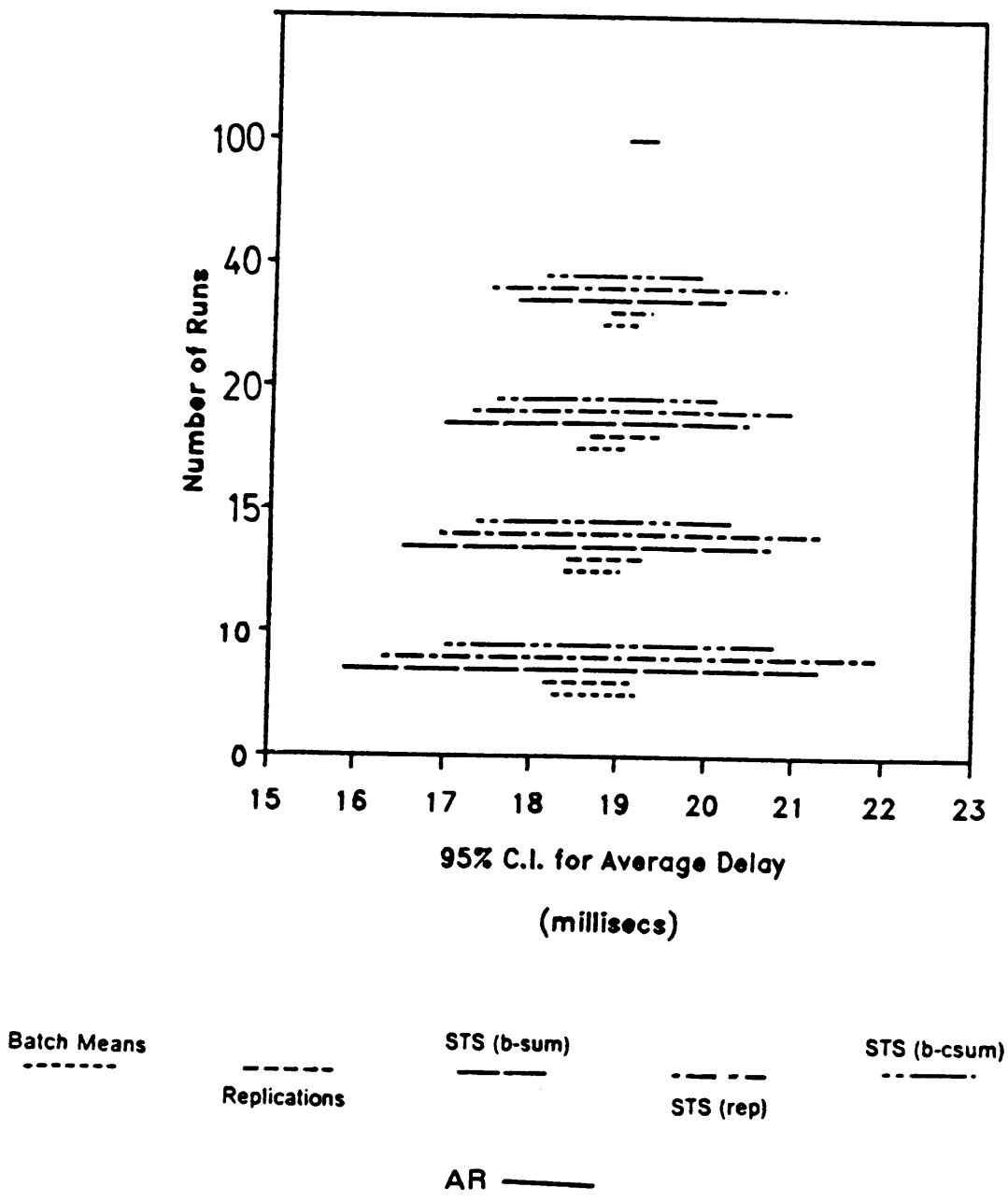


Figure 5.1. Confidence Interval Plots for the Methods of Altered Replications, Batch Means, Replications and Standardized Time Series

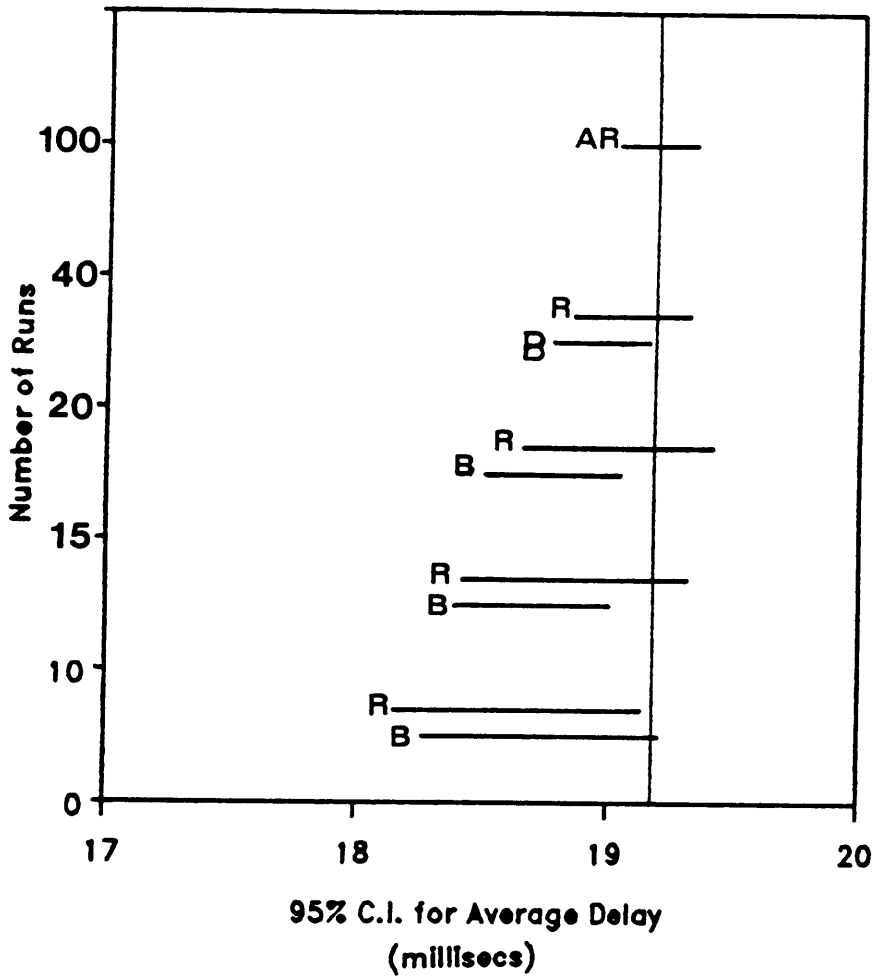


Figure 5.2. Confidence Interval Plots for the Methods of Altered Replications, Batch Means and Replications

5.1.3 Choice of Method for Further Experimentation

Section 5.1.2 indicates that the method of batch means gives the smallest half-width for average delay. The method of batch means and the standardized time series using batch means for data collection result in the lowest processing time. Due to STS giving large confidence intervals it is not the preferred method for further experimentation. Figure 5.2 illustrates the confidence interval plots for batch means and replications for the different values of k. From the plots it is evident that the performance of replications over batch means is not significantly better. In addition the cost of using batch means versus replications is reduced by more than half!. Hence the comparative analysis of the results obtained for the 11-node DHLAN using the three methods (batch means, replications and standardized time series) indicates that batch means is the most preferred method for further DHLAN experimentation.

5.1.4 Comparative Study of Batch Means and Standardized Time Series

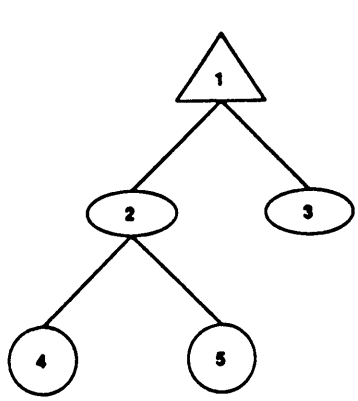
Schruben [Schruben 1983, p. 1102] compared the classical confidence interval estimator used in simulation with the four new estimators based on standardization of a time series for three strictly stationary simulation models. He observed that for large batches/replications the confidence intervals all tend to have coverages near the desired confidence coefficient. Goldsman and Schruben [Goldsman and Schruben 1984, p.1219] further studied the classical and the STS interval estimators. A study of a stationary process revealed that these new interval estimators show asymptotic properties that strictly dominate the classical estimator when used with data from independent simulation replications or batched means from a single run. In particular, the ratio of the expected interval half-width for the combined classical-sum-interval estimator to the expected half-width for the classical confidence interval estimator is < 1. For batches or replications ≤ 10, this ratio is < 0.92 (approximately).

For the 11-node DHLAN, with transition time of 300.0 milliseconds and 45000 observations in each batch/replication the ratio of interval half-width for batch means/replications to the interval half-width for STS is greater than 1 for all values of k (Table 5.2). Since the ratios are not less than 1 for the 11-node DHLAN further investigation is done with the networks comprised of 5, 11 and 15 nodes using the method of batch means and STS(b-csum). The networks are shown in Figure 5.3.

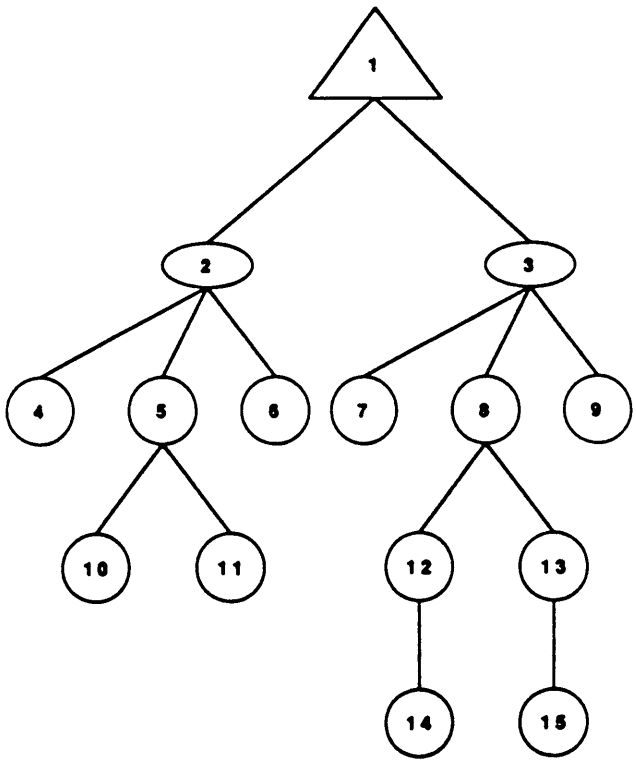
Long output series tend to yield confidence intervals with high coverage frequencies. Therefore each of the above DHLAN is run for 15 batches with a small batch size of 15000 observations. As earlier stated [section 3.1.2] a change in the traffic conditions effects a transition in the DHLAN. This causes buffering of messages at higher level nodes and reassignment of link capacities (with a consequent change in service times). This turbulent behavior during transition causes standardized times series to give higher expected confidence interval half-widths in an attempt to maintain coverage [Schruben 1983]. Since a configuration change introduces a transient which is counter to the steady-state assumption the experiment is repeated with varying transition times of 100, 300 and 600 milliseconds. The ratio of interval half-width for STS to the half-width for batch means (STS/BM) are printed at intervals of 5, 10 and 15 batches for each of the three DHLANs. Tables 5.5, 5.6 and 5.7 show the results for the 5-, 11- and 15-node dynamic hierarchy networks respectively.

The static LAN is free from the effects of transition. Hence the ratio of confidence interval half-width for STS to the half-width for batch means (STS/BM) should be less than 1. The 11-node static LAN with no traffic changes is run for 15 batches with small to large batch sizes of 4000, 15000 and 30000 observations respectively. Table 5.8 shows the results obtained for the static LAN. The standardized time series performs better than batch means for smaller batch sizes and fewer number of batches. Therefore STS produces the advantages reported by Goldsman and Schruben [Goldsman and Schruben 1984] with regard to the sampling efficiency (information derived per unit) when dynamic reconfigurability is precluded.

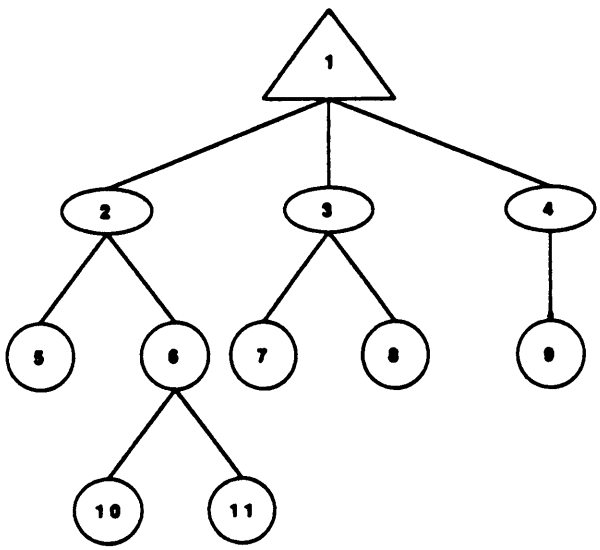
The experimental results for the dynamic hierarchy network indicate that the standardized time series method is sensitive to stationarity and tends to produce wide intervals to



5-Node DHLAN



15-Node DHLAN



11-Node DHLAN

Figure 5.3. Experimental Networks for Comparing Batch Means and Standardized Time Series

maintain coverage. The dynamic hierarchy network is a fairly complex model with varying system behavior. The transition period affects the steady-state assumption and the model tends to move away from stationary conditions. Therefore the STS may or may not always give smaller half-width estimates than those obtained for the classical confidence interval estimators. Based on the results for the static LAN, one may conclude that for a stationary process standardized time series performs better than batch means for a relatively small batch size and fewer number of batches. Where stationarity is questionable the standardized time series tends to give wide confidence intervals, hence the ratio of the expected interval half-widths for STS to the half-width for the classical confidence interval estimation techniques may or may not be less than 1.

Table 5.5. STS/BM Ratio of Expected Half-Widths for 5-Node DHLAN

Number of Batches	Transition Time in milliseconds 100	300	600
5	1.2375	0.9277	1.6266
10	1.8027	0.8284	2.6063
15	1.7248	0.8406	2.5316

Table 5.6. STS/BM Ratio of Expected Half-Widths for 11-Node DHLAN

Number of Batches	Transition Time in milliseconds 100	300	600
5	1.9529	1.3451	0.6112
10	2.9439	1.5824	0.7502
15	2.2353	1.5918	0.7764

Table 5.7. STS/BM Ratio of Expected Half-Widths for 15-Node DHLAN

Number of Batches	Transition Time in milliseconds 100	300	600
5	1.4503	0.9152	0.9503
10	1.8284	1.1384	1.2447
15	1.5578	1.2198	1.2301

Table 5.8. STS/BM Ratio of Expected Half-Widths for 11-Node Static LAN

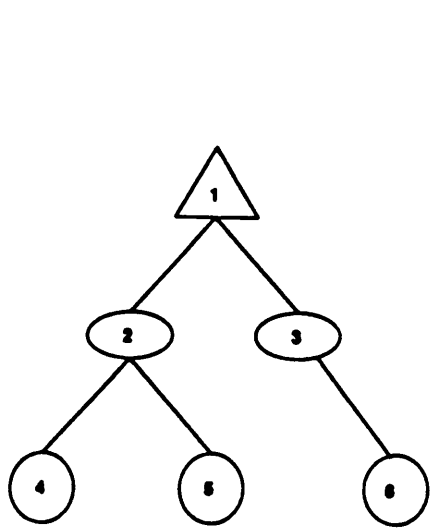
Number of Batches	Batch Size 4000	15000	30000
5	0.8548	0.7787	0.9282
10	0.9136	0.9950	0.9052
15	0.9595	1.0287	1.0455

5.2 Comparative Analysis of Dynamic Hierarchy Versus Static LANs

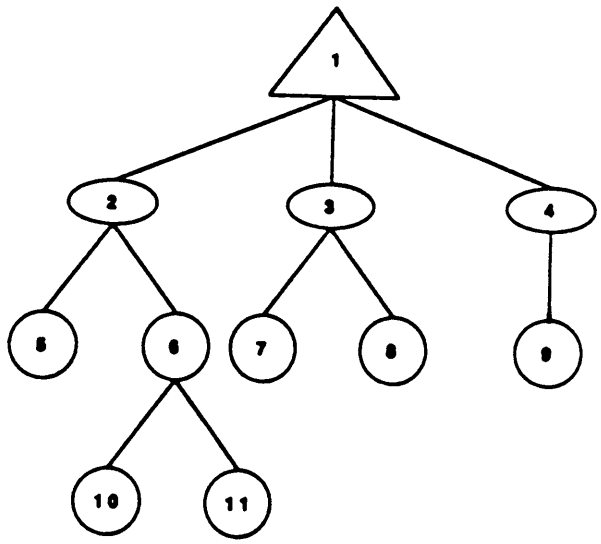
A comparative cost and benefits analysis of DHLANs and their static counterparts is essential for an initial assessment of the dynamic reconfigurability concept. The 5- and 15- node networks earlier employed for the comparison of methods are not used for the LAN comparison in an attempt to use different dynamic hierarchy networks. Networks of sizes 6, 11 and 16 nodes are selected for the comparison study. These networks represent small, medium and large sized networks respectively and are shown in Figure 5.4. The two LAN architectures are compared with respect to the key network performance measure average packet delay (termed as average delay).

For each network the experiment consists of making 10 runs (batches). Each run consists of 30000 observations and varying traffic conditions with a particular reconfiguration length (transition time). All traffic conditions exist for approximately equal time periods in each run (batch). The above is repeated using different transition times. The 6-node DHLAN is run under scenarios A, C, D and E for 3 reconfigurations and under scenario A and C for 1 reconfiguration. The 11-node DHLAN is run under scenarios B, C, D and E for 3 reconfigurations and under scenario B and D for 1 reconfiguration. The 16-node DHLAN is run under scenarios B, C, D and E for 3 reconfigurations and under scenario B and C for 1 reconfiguration. The scenarios A, B, C, D, and E are defined in section 4.1.5. The static counterparts of the 6-, 11-, and 16- node DHLANs are run under identical conditions except for the reassignment of service times when traffic changes occur. The link service times of a static network is the average of the link service times assigned to its corresponding DHLAN under different traffic conditions. The traffic matrices under the different scenarios and the transition sets for the networks are given in Appendix B.

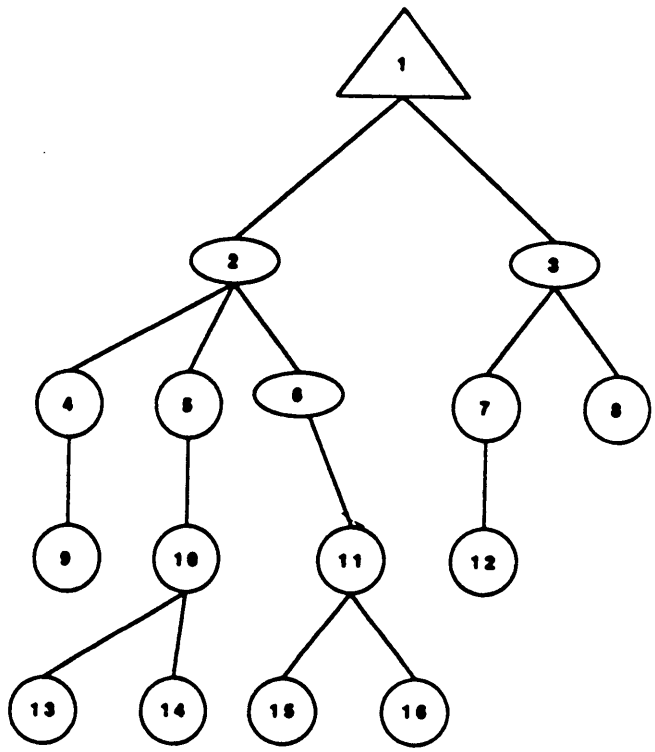
Tables 5.9-5.11 give the results obtained from experimentation with the DHLANs and their corresponding static LANs.



6-Node DHLAN



11-Node DHLAN



16-Node DHLAN

Figure 5.4. Experimental Networks for Performance Comparison of DHLAN and Static Architectures

5.2.1 Summary

A cost and benefits analysis of selected DHLANs against their static counterparts shows that DHLANs are clearly superior over their static LANs with respect to the mean packet transmission delay. The superiority becomes more obvious as the network size increases. For 3 reconfigurations, 6-, 11- and 16-node static networks show an increase in mean delay by a factor of 1, 8 and 11 respectively and by 1, 5 and 5 in the case of a single reconfiguration!. The wide confidence limits for average delay in the static case for the 11- and 16- node networks show that in contrast to the DHLAN the capability of the static LAN to handle different traffic conditions deteriorates with increasing network size. Since the static LAN is assigned fixed capacities throughout the network operation, the capacities may or may not be adequate to service different demands upon their respective links.

With reassignment of capacities, the DHLAN can handle different kinds of traffic conditions more cost effectively and efficiently than its static counterpart. Cost effectiveness is apparent in that

1. reassignment of capacity leads to better link utilization at no extra cost. A link not requiring high capacity for a particular traffic condition may distribute part of its capacity to other links, thereby enabling them to better serve traffic demands.
2. low message traffic may not require the total present network capacity hence reassignment of capacities leads to cost savings as the new total capacity is less than the previous total network capacity.

Greater efficiency of the DHLAN over the static LAN is illustrated by its handling of the queues at the ACS nodes under different traffic conditions during a simulation run. In contrast to the static LAN, the DHLAN avoids congestion under varying traffic demands and is able to service the packets without the formation of excessive queues at various nodes. Therefore the DHLAN prevents the dramatic rise in the average transmission delay with changing traffic conditions and maintains its service level in contrast to the static LAN.

Table 5.9. Average Delay Confidence Limits for Dynamic Networks

Number of Nodes	Reconfigurations = 3 Transition Time (milliseconds) 100	50	Reconfigurations = 1 Transition Time (milliseconds) 300	50
6	(10.087,10.486)	(9.904,10.115)	(9.852,10.131)	(9.234,9.374)
11	(18.626,19.571)	(17.765,18.371)	(22.429,23.554)	(18.765,19.511)
16	(26.333,29.500)	(23.168,26.314)	(22.435,25.955)	(19.543,22.320)

Table 5.10. Average Delay Confidence Limits for Static Networks

Number of Nodes	Traffic Changes = 3	Traffic Changes = 1
6	(10.301,10.634)	(10.453,10.769)
11	(130.022,181.533)	(86.486,125.328)
16	(155.491,372.951)	(27.946,191.463)

Table 5.11. Performance of Dynamic and Static Networks With Respect to Average Delay

Num. of Nodes	Num. of Reconfig.	Transition Time milliseconds	Average Delay (milliseconds) DYNAMIC	STATIC
6	3	100	10.286	10.468
		50	10.010	
6	1	300	9.992	10.611
		50	9.304	
11	3	100	19.099	155.778
		50	18.068	
11	1	300	22.992	105.907
		50	19.138	
16	3	100	27.916	264.221
		50	24.741	
16	1	300	24.195	109.704
		50	20.932	

5.3 Performance Study of Dynamic Hierarchy Networks

The clear superiority of the DHLAN architecture over the static architecture for the key network parameter average delay leads to an investigation of the DHLAN "sensitivity" to re-configuration. The experiment results obtained for the 6-, 11-, and 16-node DHLANs in the comparative study are used for performance study. In addition to average delay, the effects of dynamic reconfigurability on the throughput and network delivery ratio characteristics are studied. The above parameters are defined in section 4.2.1.

Tables 5.9, and 5.11-5.14 summarize the results obtained from experimentation with the DHLANs. The DHLAN characteristics with respect to network size, reconfiguration length and reconfiguration frequency are discussed below.

5.3.1 Network Size

The average time taken by a packet to reach its destination increases with size of the network. This is explained by the fact that the mean number of hops required by a packet is directly proportional to the size of the network. An increase in network size implies that on an average a packet has to visit more nodes before reaching its final destination, hence an increase in mean delay.

The network throughput defined as the number of packets delivered to their final destination per millisecond increases with an increase in network size. The total number of messages generated per millisecond in a network is higher than the number generated for a comparatively smaller sized network. If a network is not congested then under some upper bound a higher message traffic implies a higher rate of packets received. The total number of messages generated per millisecond in the 6-, 11-, and 16-node networks are about 80, 150 and 215 packets/millisecond. The increase in throughput from 0.5 to 1.6 corresponding to an increase in size from 6 to 16 nodes supports this idea.

The network delivery ratio defined as the ratio of the number of packets delivered to the number of packets generated in the network is approximately 1 for all networks. This implies that the process of network reconfiguration and buffering of messages at higher level nodes during transition does not create a bottleneck. Reconfigured networks (small or large) have the capability to recover from transition overhead and service the incoming and queued packets fast enough avoiding congestion thus maintaining the level of service under all scenarios.

5.3.2 Reconfiguration Length

A reduction in reconfiguration length (transition time) always reduces the mean packet delay. This is obvious since a small transition time implies buffering of fewer messages at ACS nodes during transition, hence lesser overhead when transition is over. Transition time therefore should be kept as low as possible. In the case of 3 reconfigurations a reduction in transition time by half causes reduction in mean delay estimates by a percentage of 2.68, 5.40 and 11.37 for the 6-, 11-, and 16-node networks respectively. This shows that under approximately identical traffic scenarios the DHLAN is more sensitive to reconfiguration length as the network size increases.

In the case of one reconfiguration a reduction in transition time by 1/6 causes reduction in average delay by a percentage of 6.89, 16.26 and 13.49 for the 6-, 11-, and 16-node DHLANs respectively. The initial traffic scenarios for the 6-, and 16-node networks changes to the scenario C that exhibits heavy locality amongst neighboring nodes while that for the 11-node DHLAN changes to scenario D where more packets travel across the apex. Since the scenario D induces a heavier penalty on average delay than the scenario C, the 11-node DHLAN shows greater "sensitivity" to reduction in reconfiguration length than the 16-node DHLAN for one reconfiguration. This implies that sensitivity to a reduction in reconfiguration length increases under traffic conditions that incur heavy load on the DHLAN links.

5.3.3 *Reconfiguration Frequency*

The 6- and 16-node DHLANs show an increase in mean delay estimates by a factor of 1.03 and 1.15 (i.e. less than 20% increase) respectively in the case of 3 reconfigurations with transition time of 100 milliseconds and 1 reconfiguration with transition time of 300 milliseconds. This shows that the DHLAN does not accumulate transition overhead by a factor corresponding to the number of transitions. Since the 11-node DHLAN is run under heavy traffic load conditions for 1 reconfiguration, the estimates for mean delay for 3 reconfigurations with transition time of 100 milliseconds are less than the estimates for 1 reconfiguration with transition time of 300 milliseconds. This shows that under certain circumstances the DHLAN can give improved performance for higher reconfiguration frequencies.

This discussion is also observed to hold for the case of 3 reconfigurations with transition time of 50 milliseconds and 1 reconfiguration with transition time of 50 milliseconds. However, it should be noted that the increase in mean delay estimates in this case for the 6- and 16-node DHLANs are by a factor of 1.08 and 1.18 respectively. Therefore, under the same reconfiguration length, the DHLAN is more sensitive to reconfiguration frequency.

5.3.4 *Summary*

The performance study of the selected DHLANs reveals that the average time taken by a packet to reach its destination increases with an increase in network size. Network throughput also increases with size of the network. Reconfigured networks (small and large) with the reassignment of new link capacities (service times) have the capability to recover from transition overhead and maintain the level of service under the different traffic conditions.

A reduction in reconfiguration length by half reduces the mean transmission delay by three percent for a 6-node network and by twelve percent for a 16-node network. Reconfiguration length should therefore be kept as low as possible to optimize network performance with respect to delay. The sensitivity to reduction in reconfiguration length increases under

traffic conditions that incur heavy load on the DHLAN links. The DHLAN does not accumulate transition overhead by a factor corresponding to the reconfiguration frequency.

Table 5.12. Throughput Confidence Limits for Dynamic Networks

Num. of Nodes	Reconfigurations = 3 Transition Time (milliseconds) 100	50	Reconfigurations = 1 Transition Time (milliseconds) 300	50
6	(0.508,0.512)	(0.506,0.510)	(0.496,0.499)	(0.495,0.500)
11	(1.051,1.062)	(1.052,1.058)	(1.048,1.054)	(1.051,1.058)
16	(1.610,1.620)	(1.611,1.623)	(1.505,1.518)	(1.504,1.515)

Table 5.13. Network Delivery Ratio Confidence Limits for Dynamic Networks

Num. of Nodes	Reconfiguration = 3 Transition Time (milliseconds) 100	50	Reconfiguration = 1 Transition Time (milliseconds) 300	50
6	(0.999,0.999)	(0.999,0.999)	(0.999,0.999)	(0.999,1.000)
11	(0.999,0.999)	(0.998,0.999)	(0.998,0.999)	(0.998,0.999)
16	(0.998,0.999)	(0.998,0.999)	(0.999,0.999)	(0.999,0.999)

Table 5.14. Performance of Dynamic Networks With Respect to Throughput and Network Delivery Ratio

Num. of Nodes	Num. of Reconfig.	Transition Time milliseconds	Throughput pkts/millisecond	Network Delivery Ratio
6	3	100	0.510	0.999
		50	0.508	0.999
6	1	300	0.497	0.999
		50	0.497	0.999
11	3	100	1.056	0.999
		50	1.055	0.998
11	1	300	1.051	0.999
		50	1.055	0.999
16	3	100	1.615	0.999
		50	1.617	0.998
16	1	300	1.511	0.999
		50	1.509	0.999

6.0 CONCLUSIONS AND FUTURE RESEARCH

The important results and conclusions of this research are summarized in section 6.1. Section 6.2 gives the recommendations for future research.

6.1 Important Results and Conclusions

6.1.1 Credibility of the DHLAN Simulation Model

The credibility of the simulation model has been established with emphasis on model verification throughout the investigation. Model validation has been accomplished by assessing the correctness of the program representation and assuring that under specified parameter and input conditions, the model behavior conforms with the predictions.

6.1.2 Comparison of the Three Methods

A comparative study of the standardized time series (STS) method and the method of batch means shows that in agreement with the prior results reported by Goldsman and Schruben [Goldsman and Schruben 1984], STS produces the advantages with regard to the sampling efficiency (information derived per sample unit) for the static architecture for a relatively small batch size and fewer number of batches. However, in the case of the dynamically reconfigurable architectures STS exhibits greater sensitivity to the non-stationarity induced by reconfiguration. As a result STS gives higher confidence interval half-widths than the classical interval estimators in an attempt to preserve coverage.

The method of batch means yields smaller variance estimates than the method of replications and is more cost effective since the DHLAN has a long transient period. Therefore batch means becomes the preferred choice for further experimentation with the DHLAN simulation model.

6.1.3 Comparison of the DHLAN and Static LAN Architectures

A comparison of the two network architectures shows that under high traffic variability the dynamic hierarchy architecture is clearly superior to the static architecture with respect to average packet transmission delay for the cases tested using the described model. This superiority becomes more obvious as the network size increases. With reassignment of link capacities (new service times), the DHLAN has a much better service potential and can handle different traffic conditions more cost effectively and efficiently than its static counterpart.

6.1.4 Performance Analysis of DHLANs

A limited performance study of selected DHLANs shows that the average time taken by a packet to reach its destination increases with an increase in the network size. The network throughput also increases with network size. Reconfigured networks (small and large) have the capability to recover from transition overhead and maintain the level of service under different traffic conditions.

A reduction in reconfiguration length by half reduces the mean transmission delay by three percent for a six-node network and by twelve percent for a sixteen-node network. Reconfiguration length should therefore be kept as low as possible to optimize network performance with respect to delay. Mean network delay appears to exhibit sensitivity to an interaction effect between reconfiguration period length and the level of message demand. The DHLAN does not accumulate transition overhead commensurate with the reconfiguration frequency. An increase in reconfiguration frequency by three increases the average delay by less than twenty percent for the networks tested.

6.2 Recommendations for Future Research

As a basis for future research, the work reported in this thesis has verified the DHLAN model and established that the dynamic hierarchy concept warrants further evaluation. The following issues need to be addressed:

1. The reconfiguration capability needs to be enhanced. All nodes should have the capability to "sense" the traffic change and initiate a transition. The ACS nodes should have further capability to abort a transition, if need be.
2. Active and inactive links should be incorporated in the model to render the advantages of apex transition (survivability and reliability) more visible.
3. The network model should incorporate both nodal and link service times and also generate local messages.
4. The simulation results need to be compared with an equivalent analytical model so that future research can be conducted using the analytical model to avoid the high simulation costs.
5. Issues relating to message acknowledgement, error control and alternate routing paths in the case of active and inactive links need to be addressed and incorporated in the model.

BIBLIOGRAPHY

Adrion, R.W., M.A. Branstad, and J.C. Cherniavsky (1982), "Validation, Verification and Testing of Computer Software," *Computing Surveys,* Vol. 14(2), June, pp. 159-192.

Anderson, R.R. (1972), "Simulated Performance of a Ring-Switched Data Network", *IEEE Trans. Commun.,* Vol. Com-20(3), June, pp. 576-591.

Anderson, T.W. and S.L. Sclove (1978), *An Introduction to the Statistical Analysis of Data,* Houghton Mifflin Comopany, Boston.

Balci, O. (1985), "Requirements For Model Development Environments," Technical Report SRC-85-006, Systems Research Center and Department of Computer Science, Virginia Tech, Blacksburg, Virginia, February.

Balci, O. (1986), "Guidelines For Successful Simulation Studies Part II," Technical Report SRC-86-002, Systems Research Center and Department of Computer Science, Virginia Tech, Blacksburg, Virginia, September.

Balci, O. (1987), "Credibility Assessment of Simulation Results: The State of the Art", *Proceedings of the Conference on Methodology and Validation,* (1987 ESC, Orlando, Fla., April 6-9), Published as Simulation Series 19, 1 (January 1988), 19-25. SCS, San Diego, Calif..

Bhat, U.N. and R.E. Nance (1987), "A Queueing Network Analysis of Dynamic Reconfigurability in an Hierarchical Information Network", Technical Report SRC 86-007, March.

Chen, B.M. and R.G. Sargent (1987), "Using Standardized Time Series to Estimate Confidence Intervals for the Difference Between Two Stationary Stochastic Processes", *Operations Research,* Vol. 35(3), May-June, pp. 428-436.

Conway, R.W. (1963), "Some Tactical Problems in Digital Simulation," *Management Science,* Vol. 10(1), October, pp. 47-61.

Crane, M.A., and D.L. Iglehart (1975), "Simulating Stable Stochastic Systems: III. Regenerative Proceses and Discrete-Event Simulation", *Operations Research,* Vol. 23(1), January-February, pp. 33-45.

Dahlberg, A. (1983), "A Performance Study of Fast Loop Networks," *Local Computer Networks,* October, pp. 37-43.

Derrick, E.J. and R.E. Nance (1986), "Local Area Networks and the Dynamic Hierarchy: A Tutorial," Technical Report TR-86-16, Systems Research Center and Department of Computer Science, Virginia Tech, May.

Dhadesugoor, V.R. and P.V. Autenried (1986), "Real-Time Simulation of a Prioritized Dynamic Access Protocol for Local Area Networks", Dept. of Computer Science, Stevens Institute of Technology, Technical Report # CS 8607, March.

Emshoff, J.R. and R.L. Sisson (1970), *Design and Use of Computer Simulation Models,* MacMillan, New York.

Fenig, B. and M. Myron (1979), "A Generic Program for the Modeling and Simulation of Distributed Computer System Networks", *Local Computer Networks,* October, pp. 97-103.

Fishman, G.S. (1973), "Statistical Analysis for Queueing Simulations," *Management Science,* Vol. 20(3), November, pp. 363-369.

Fishman, G.S. (1974), "Estimation in Multiserver Queueing Simulations," *Operations Research,* Vol. 22(1), January-February, pp. 72-78.

Fishman, G.S. (1978), *Principles of Discrete Event Simulation,* Wiley-Interscience, New York.

Gaspar, A. and P. Lamm (1978), "A Simulated Data Communication Network," *Computer Communications Review (ACM),* Vol. 8(4), October, pp. 19-29.

Glass, R.L. (1979), *Software Reliability Guidebook,* Prentice Hall, Englewood Cliffs, New Jersey. Goldsman, D. and L. Schruben (1984), "Asymptotic Properties of Some Confidence Interval estimators For Simulation Output Analysis", *Management Science,* Vol. 30(10), October, pp.1217-1225.

Gordon, G. (1978), *"System Simulation,"* Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632.

Hansen, A. (1978), "The EIN Network Simulation (European Information Network)", In *Computer Networks and Simulation,* S. Schoemaker (ed.), North-Holland Publishing Company, Amsterdam, pp. 249-256.

Hogrefe, D. and S. Zorn (1986), "Simulation of SDL Specified Models," *Computer Networks and Simulation III,* S. Schoemaker (ed.), North-Holland Publishing Company, Amsterdam, pp. 103-122.

Katkin, R.D. and J.G. Sprung (1981), "Simulating a Cable Bus Network in a Multi-Computer and Large Scale Applications Environment", *Local Computer Networks,* October, pp. 82-90.

Labetoulle, J. (1977), "A Homogenous Computer Network : Analysis and Simulation", *Computer Networks,* Vol. 1(4), May, pp. 225-240.

Law, A.M. (1977), "Confidence Intervals in Discrete Event Simulation: A Comparision of Replication and Batch Means", *Nav. Res. Logist. Q.,* Vol. 24, pp. 667-678.

Law, A.M. and D.W. Kelton (1982), *Simulation Modeling and Analysis,* McGraw-Hill Book Company, New York.

Leh, J.W. (1979), "Simulations of Local Computer Networks," *Local Computer Networks,* October, pp. 56-66.

Moose, R.L. Jr. (1983), "Link Capacity Assignment in Dynamic Hierarchical Networks For Real-Time Applications", M.S. Thesis, Department of Computer Science, Virginia Tech, October.

Moose, R.L. Jr. and R.E. Nance (1987), "Link Capacity Assignment in Dynamic Hierarchical Networks," Technical Report SRC 87-011, Systems Research Center and Department of Computer Science, Virginia Tech, July - to be published.

Nagappan, S. (1986), "Protocol design and Analysis For A Dynamic Hierarchical Local Area Network", M.S. Thesis, Department of Computer Science, Virginia Tech, June.

Nance, R.E. (1979), "Distributed Computing: An Overview," Combat Systems Department Presentation, Naval Surface Weapons Center, Dahlgren, Virginia, December.

Nance, R.E. (1981), "Model Representation in Discrete Event Simulation: The Conical Methodology", Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia, March.

Nance, R.E. (1983), "A Tutorial View of Simulation Model Development," *Proceedings of the 1983 Winter Simulation Conference* (Arlington, Virginia), IEEE, New Jersey, December, pp. 325-331.

Probert, R.L. (1982), "Optimal Insertion of Software Probes in Well-Delimited Programs", *IEEE Transactions on Software Engineering SE-8*, # 1, January, pp. 34-42.

Razouk R.R. (1986), "Modeling and Simulation of Data Communication Networks using SARA", *Computer Networks and Simulation III*, S. Schoemaker (ed.), North-Holland Publishing Company, Amsterdam, pp. 177-200.

Reames, C.C. and M.T. Liu (1976), "Design and Simulation of the Distributed Loop Computer Network (DLCN)", *Proc. of the 3rd Annual Symposium on Computer Architecture*, Vol. 4(4), January, pp. 124-127.

Rogeberg, T. (1978), "TETRASIM: A Program System for the Simulation of Telephone Networks", *Computer Networks and Simulation*, S. Schoemaker (ed.), North-Holland Publishing Company, Amsterdam, pp. 195-226.

Schruben, L. (1983), "Confidence Interval Estimation Using Standardized Time Series", *Operations Research*, Vol. 31(6), November-December, pp. 1090-1108.

Schwartz, M. (1977), *Computer Communication Network Design and Analysis*, Prentice Hall, Englewood Cliffs, New Jersey.

Hammarskjoldsvei, D. (1986), *"SIMULA Programmmer's Guide IBM 370"*, Simula a.s., N-0535, Oslo.

Van Horn, R.L. (1971), "Validation of Simulation Results," *Management Science*, Vol. 17(5), May, pp. 247-258.

Vincent, S.G., and A.M. Law (1983), *UNIFIT: An Interactive Computer Package For Fitting Probability to Observed Data*, User's Guide, Simulation Modeling and Analysis Company, Tucson, Arizona.

Watson, B. W. (1979), "Simulation Study of the Traffic Dependent Performance of a prioritized, CSMA Broadcast Network", *Local Computer Networks*, October, pp. 67-74.

Wolf, J.J., B.W. Weide and M.T. Liu (1979), "Analysis and Simulation of the Distributed Double Loop Network (DDLCN)", *IEEE Computer Networking Symposium*, pp. 82-89.

Appendix A. DHLAN PROGRAMMED MODEL AND EXPERIMENT DESIGN VERIFICATION

Note: Comments are inserted between /* */ to explain the various entries in the output. Only batch 1 entries are commented. Subsequent batches can be clearly understood based on the information provided for the initial batch.

```
/* INPUT DATA */
/***************/

/* Traffic matrix under scenario A for 6 node DHLAN. */
   17.0000     0.0000  0 20 20 20 20 20
   16.8800     0.7200 20  0 20 20 20 20
   15.0000     0.6400 20 20  0 20 20 20
    9.3800     0.4000 20 20 20  0 20 20
    9.3800     0.4000 20 20 20 20  0 20
    9.3800     0.4000 20 20 20 20 20  0

/* Next transition takes place after 100.0 milliseconds. The initiating
node being node(5) and the new apex to be is node(2). */
  100.0000   5   2

/* Traffic matrix under scenario C for 6 node DHLAN */
   15.3200     0.4800  0 40 30 10 10 10
   16.0000     0.0000 25  0 15 25 25 10
   13.4000     0.4200 30 30  0  5  5 30
   10.5300     0.3300 20 50  5  0 20  5
   10.5300     0.3300 20 50  5 20  0  5
   10.2100     0.3200 25 20 45  5  5  0

/* Next transition takes place after 100.0 milliseconds. The initiating
node being node(6) and the new apex to be is node(3). */
  100.0000   6   3

/* Traffic matrix under scenario D for 6 node DHLAN */
   18.7500     0.8000  0 20 40  5  5 30
   16.6400     0.7100 20  0 35 10 10 25
   19.0000     0.0000 25 25  0 15 15 20
    7.2700     0.3100 10 20 40  0 10 20
    7.2700     0.3100 10 20 40 10  0 20
   10.0800     0.4300 20 20 30 15 15  0

/* Next transition takes place after 100.0 milliseconds. The initiating
node being node(3) and the new apex to be is node(1). */
  100.0000   3   1

/* Traffic matrix under scenario E for 6 node DHLAN */
   18.0000     0.0000  0 20 20 20 20 20
   17.1400     0.5600 50  0  5 20 20  5
   13.1600     0.4300 50 10  0  5  5 30
    9.8000     0.3200 50 30  5  0 10  5
    9.8000     0.3200 50 30  5 10  0  5
   10.1000     0.3300 50 10 30  5  5  0

/* OUTPUT DATA */
/***************/

PROGRAM OUTPUT BEGINS FOR VERIFICATION OF THE NETWORK

USING THE METHOD OF BATCH MEANS FOR DATA COLLECTION
NETWORK PARAMETERS:
    NO OF NODES         :        6
    NO OF APEXES        :        3
    TRANSITION TIME     :    25.000 MILLISECONDS

/* Initial values for the seeds U1 and U2. */
    FIRST U1 :              212176
    FIRST U2 :                 985

/* The successor matrix for the 6 node DHLAN shown in Figure A.1a under
  scenario A */
 0 2 3 2 2 3
 0 0 0 4 5 0
 0 0 0 0 0 6
```

```
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0

/* Checking of the nodal attributes. Row(i) denotes if node(i) is
(apex, apex candidate candidate). 11 denotes node(1) is an apex node and
an apex candidate node. */
11
01
01
00
00
00


START OF TRACE MONITORING AFTER
   50 PKTS DELIVERED IN TIME   102.524

QUEUE CONTENTS FOR ALL QUEUES FOR NODES 1 TO  6
    0    1    0    0    1    0

BATCH :  1
FIRST U1 :          -1247000048
FIRST U2 :           1144989925

/* Initializations at start of data collection */
PKTS RECEIVED :           0
TOTAL DELAY   :           0.000000

/* Configuration changer activated */
TRAFFIC MTX CHANGES NOW AT TIME   100.000 MILLISECS FROM START TIME

/* New traffic conditions reflecting scenario C */
   0  40  30  10  10  10
  25   0  15  25  25  10
  30  30   0   5   5  30
  20  50   5   0  20   5
  20  50   5  20   0   5
  25  20  45   5   5   0

/* initiating node: node(5) and new apex to be: node(2) */
   5   2

/* Daemon statistics collector prints sum of the queue lengths of ACS
nodes at intervals of 100.0 milliseconds */
      100.000    0

/* New successor matrix with node(2) as the apex node. The new
hierarchical structure is shown in Figure A.1b */
 0 0 3 0 0 3
 1 0 1 4 5 1
 0 0 0 0 0 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0

/* New nodal service times corresponding to each node. The apex node
(node(2) is assigned zero service time. */
 0.4800 0.0000 0.4200 0.3300 0.3300 0.3200

/* Checking of nodal attributes after transition. Node(2) is the new
apex. */
01
11
01
00
```

```
00
00

/* Configuration changer activated */
TRAFFIC MTX CHANGES NOW AT TIME   200.000 MILLISECS FROM START TIME

/* New traffic conditions reflecting scenario D */
   0  20  40   5   5  30
  20   0  35  10  10  25
  25  25   0  15  15  20
  10  20  40   0  10  20
  10  20  40  10   0  20
  20  20  30  15  15   0

/* initiating node: node(6) and new apex to be: node(3) */
   6   3

/* Daemon statistics collector prints sum of the queue lengths of ACS
nodes at intervals of 100.0 milliseconds */
      200.000    1

/* New successor matrix with node(3) as the apex node. The new
hierarchical structure is shown in Figure A.1c. */
 0 2 0 2 2 0
 0 0 0 4 5 0
 1 1 0 1 1 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0

/* New nodal service times corresponding to each node. The apex node
(node(3) is assigned zero service time. */
 0.8000 0.7100 0.0000 0.3100 0.3100 0.4300

/* Checking of nodal attributes after transition. Node(3) is the new
apex. */
01
01
11
00
00
00

/* Configuration changer activated */
TRAFFIC MTX CHANGES NOW AT TIME   300.000 MILLISECS FROM START TIME

/* New traffic conditions reflecting scenario E */
   0  20  20  20  20  20
  50   0   5  20  20   5
  50  10   0   5   5  30
  50  30   5   0  10   5
  50  30   5  10   0   5
  50  10  30   5   5   0

/* initiating node: node(3) and new apex to be: node(1) */
   3   1

/* Daemon statistics collector prints sum of the queue lengths of ACS
nodes at intervals of 100.0 milliseconds */
      300.000   18

/* New successor matrix with node(1) as the apex node */
 0 2 3 2 2 3
 0 0 0 4 5 0
 0 0 0 0 0 6
 0 0 0 0 0 0
 0 0 0 0 0 0
```

```
 0 0 0 0 0 0

/* New nodal service times corresponding to each node. The apex node
(node(1) is assigned zero service time. */
 0.0000 0.5600 0.4300 0.3200 0.3200 0.3300

/* Checking of nodal attributes after transition. Node(1) is the new
apex. */
11
01
01
00
00
00

/* Daemon statistics collector prints sum of the queue lengths of ACS
nodes at intervals of 100.0 milliseconds */
      400.000    4
      500.000    0      600.000    0
      700.000    3

QUEUED PKTS AT EACH NODE :  0  0  0  3  2  0



TRANSITIONS OCCURED  :        3
PKTS GENERATED       :      407
RECEIVED             :      400
PACKETS IN SYSTEM    :        7
SIMULATION TIME      :         787.089800
TOTAL_DELAY          :        6069.046278 MILLISECONDS
AVERAGE DELAY        :          15.172616 MILLISECONDS
THROUGHPUT           :           0.517095 PKTS/MILLISECONDS
NET DELIVERY RATIO   :           0.982801

LAST  U1 :          1576996816
LAST  U2 :         -1656243795
BATCH :   2

FIRST U1 :          1576996816
FIRST U2 :         -1656243795

START TIME    :        889.613824


TRAFFIC MTX CHANGES NOW AT TIME   100.000 MILLISECS FROM START TIME
   0  40  30  10  10  10
  25   0  15  25  25  10
  30  30   0   5   5  30
  20  50   5   0  20   5
  20  50   5  20   0   5
  25  20  45   5   5   0
   5   2
      100.000    3
 0 0 3 0 0 3
 1 0 1 4 5 1
 0 0 0 0 0 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0.4800 0.0000 0.4200 0.3300 0.3300 0.3200
01
11
01
00
00
00
```

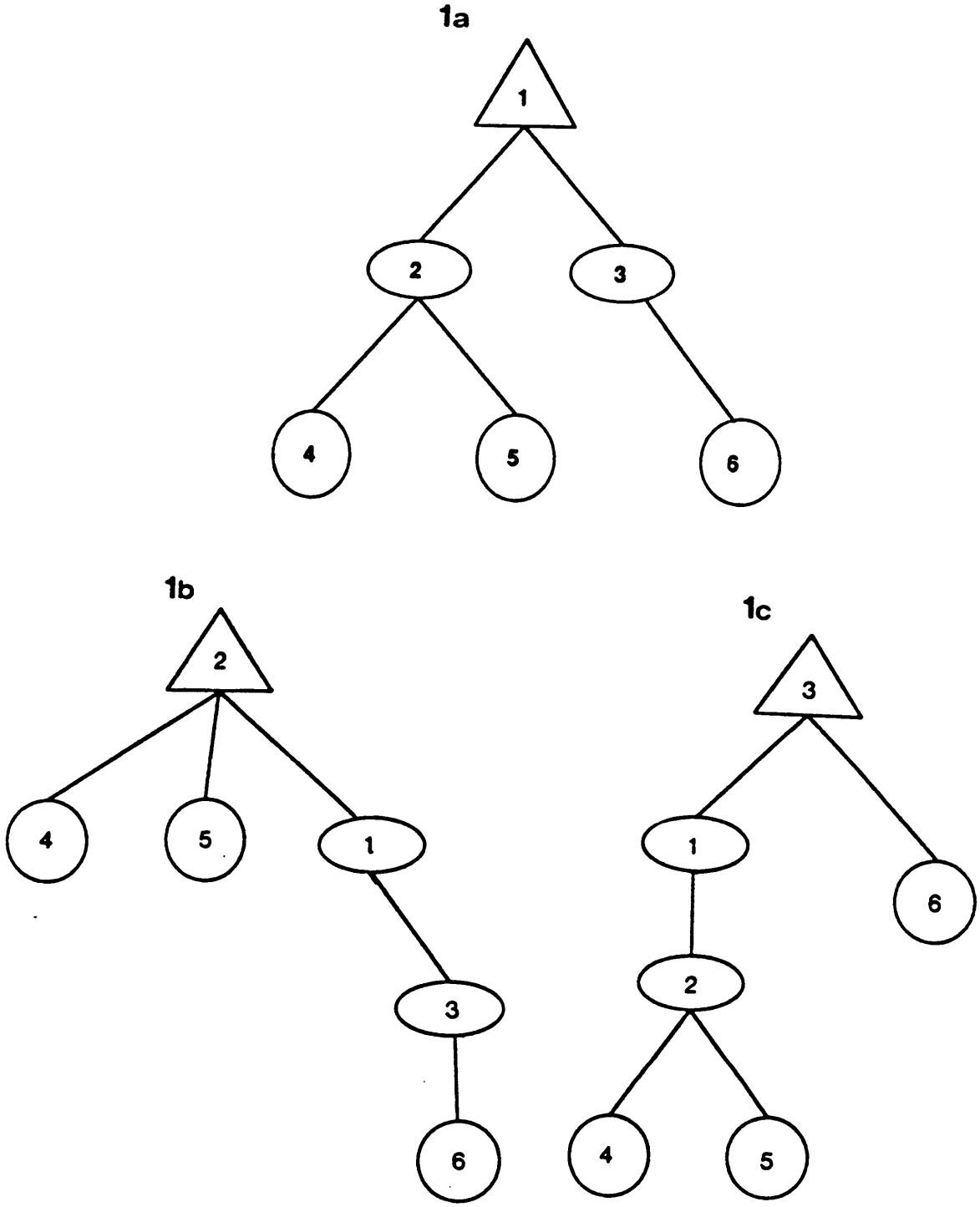


Figure A.1. The 6-Node Network Topology Under Different Scenarios

```
 TRAFFIC MTX CHANGES NOW AT TIME   200.000 MILLISECS FROM START TIME
    0  20  40   5   5  30
   20   0  35  10  10  25
   25  25   0  15  15  20
   10  20  40   0  10  20
   10  20  40  10   0  20
   20  20  30  15  15   0
    6   3
       200.000    0

  0 2 0 2 2 0
  0 0 0 4 5 0
  1 1 0 1 1 6
  0 0 0 0 0 0
  0 0 0 0 0 0
  0 0 0 0 0 0
  0.8000 0.7100 0.0000 0.3100 0.3100 0.4300
 01
 01
 11
 00
 00
 00

 TRAFFIC MTX CHANGES NOW AT TIME   300.000 MILLISECS FROM START TIME
    0  20  20  20  20  20
   50   0   5  20  20   5
   50  10   0   5   5  30
   50  30   5   0  10   5
   50  30   5  10   0   5
   50  10  30   5   5   0
    3   1
       300.000    0
  0 2 3 2 2 3
  0 0 0 4 5 0
  0 0 0 0 0 6
  0 0 0 0 0 0
  0 0 0 0 0 0
  0 0 0 0 0 0
  0.0000 0.5600 0.4300 0.3200 0.3200 0.3300
 11
 01
 01
 00
 00
 00
       400.000    1
       500.000    1      600.000    0
       700.000    0

 QUEUED PKTS AT EACH NODE :  0  0  0  0  0  1


 TRANSITIONS OCCURED  :        3
 PKTS GENERATED       :      403
 RECEIVED             :      400
 PACKETS IN SYSTEM    :        3
 SIMULATION TIME      :        724.167093
 TOTAL_DELAY          :        3742.093352 MILLISECONDS
 AVERAGE DELAY        :           9.355233 MILLISECONDS
 THROUGHPUT           :           0.556501 PKTS/MILLISECONDS
 NET DELIVERY RATIO   :           0.992556

 LAST  U1 :          1318778896
 LAST  U2 :         -1312536623
```

```
BATCH :   3

FIRST U1 :          1318778896
FIRST U2 :         -1312536623

PKTS RECEIVED :          0
TOTAL DELAY   :          0.000000
START TIME    :       1613.780917


TRAFFIC MTX CHANGES NOW AT TIME   100.000 MILLISECS FROM START TIME
   0  40  30  10  10  10
  25   0  15  25  25  10
  30  30   0   5   5  30
  20  50   5   0  20   5
  20  50   5  20   0   5
  25  20  45   5   5   0
   5   2
      100.000     0
 0 0 3 0 0 3
 1 0 1 4 5 1
 0 0 0 0 0 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0.4800 0.0000 0.4200 0.3300 0.3300 0.3200
01
11
01
00
00
00

TRAFFIC MTX CHANGES NOW AT TIME   200.000 MILLISECS FROM START TIME
   0  20  40   5   5  30
  20   0  35  10  10  25
  25  25   0  15  15  20
  10  20  40   0  10  20
  10  20  40  10   0  20
  20  20  30  15  15   0
   6   3
      200.000     0

 0 2 0 2 2 0
 0 0 0 4 5 0
 1 1 0 1 1 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0.8000 0.7100 0.0000 0.3100 0.3100 0.4300
01
01
11
00
00
00

TRAFFIC MTX CHANGES NOW AT TIME   300.000 MILLISECS FROM START TIME
   0  20  20  20  20  20
  50   0   5  20  20   5
  50  10   0   5   5  30
  50  30   5   0  10   5
  50  30   5  10   0   5
  50  10  30   5   5   0
   3   1
      300.000     0
```

```
 0 2 3 2 2 3
 0 0 0 4 5 0
 0 0 0 0 0 6
 0 0 0 0 0 0
 0 0 0 0 0 0
 0 0 0 0 0 0
 0.0000 0.5600 0.4300 0.3200 0.3200 0.3300
11
01
01
00
00
00
      400.000    0
      500.000    0      600.000    0
      700.000    2

QUEUED PKTS AT EACH NODE :  0  0  0  0  0  1


TRANSITIONS OCCURED  :        3
PKTS GENERATED       :      404
RECEIVED             :      400
PACKETS IN SYSTEM    :        4
SIMULATION TIME      :         763.675461
TOTAL_DELAY          :        4617.846013 MILLISECONDS
AVERAGE DELAY        :          11.544615 MILLISECONDS
THROUGHPUT           :           0.529021 PKTS/MILLISECONDS
NET DELIVERY RATIO   :           0.990099

LAST  U1 :          1602009744
LAST  U2 :          -455548795


ANALYSIS OF SIMULATION RESULTS FOR THE METHOD OF BATCH MEANS
------------------------------------------------------------


 6  NODE DHLAN STUDY

NUMBER OF BATCHES :   3

NO_APEX_CANDIDATES  NO_TRANSITIONS  TRANSITION_TIME
        3                    3          25.0 MILLISECS

BATCH MEANS FOR THE NETWORK PARAMETERS
**************************************

 BATCH      AVERAGE DELAY      THROUGHPUT     NET DELIVERY RATIO
              MILLISECS


   1           15.173            0.517              0.98280
   2            9.355            0.557              0.99256
   3           11.545            0.529              0.99010


 OVERALL MEAN FOR THE THREE NETWORK PARAMETERS
************************************************

 AVERAGE DELAY      THROUGHPUT     NET DELIVERY RATIO
    12.024           0.53421           0.98849
```

AUTOCOVARIANCE FUNCTION
****

	NETWORK PARAMETERS		
	AVERAGE DELAY	THROUGHPUT	NET DELIVERY RATIO
R(0)	5.75530E+00	2.72256E-04	1.71616E-05
R(1)	-3.56157E+00	-2.48552E-04	-8.28472E-06
R(2)	-1.50981E+00	8.87200E-05	-9.17294E-06

CORRELATION BETWEEN THE NEIGHBOURING BATCH MEANS
****

RHO	NETWORK PARAMETERS		
	AVERAGE DELAY	THROUGHPUT	NET DELIVERY RATIO
1	-4.12555E-01	-6.08623E-01	-3.21832E-01
2	-8.74447E-02	1.08623E-01	-1.78168E-01

TVALUE 4.303

0.95 CONFIDENCE INTERVAL ESTIMATES
****

	NETWORK PARAMETERS		
	AVERAGE DELAY	THROUGHPUT	NET DELIVERY RATIO
LOWER	4.725	0.484	0.975881
UPPER	19.324	0.584	1.001090
HALF WIDTH	7.29945E+00	5.02048E-02	1.26048E-02
VARIANCE	2.87765E+00	1.36128E-04	8.58079E-06

ZYQ994 END OF SIMULA PROGRAM EXECUTION AT 10:26:09.24 EXECUTION TIME 1.08 SEC. RETURN CODE IS #00000000
ZYQ994 00002 STORECOLLAPSES, USING 0.11 SECONDS CPU TIME

Appendix B. EXPERIMENTAL TRAFFIC MATRICES AND TRANSITION SETS

A description of the different scenarios A-E is given in section 4.1.5. The entries in the traffic matrices for the selected test networks signify the following:

Arrival Rate : Packet arrival rate at a node in packets/millisec.

Service time : Link service time represented by a node in millisecs.

Traffic Matrix : Each entry relects the average rate of message flow between that node-pair in packets/second.

The traffic matrices for 5-, 11-, and 15-node networks (Tables B.1-B.6) have been used for comparing the methods of batch means and standardized time series.

The traffic matrices and transition sets for the 6-, 11-, and 16-node networks (Tables B.7-B.27) have been used for comparing the dynamic and static architectures and for performance analysis of the DHLANs. Only one traffic matrix for each of the 6-, 11-, and 16-node static LANs has been given. The remaining traffic matrices for the static LANs correspond to the traffic matrices of their respective DHLANs. The only difference being the service times. This column is absent for the remaining traffic matrices of the static LANs since the service times are assigned once for the entire simulation (in the first traffic condition).

Table B.1. Traffic Matrix Under Scenario A for 5-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix				
			1	2	3	4	5
1	14.00	0.000	0	25	25	25	25
2	13.33	0.600	25	0	25	25	25
3	8.89	0.400	25	25	0	25	25
4	8.89	0.400	25	25	25	0	25
5	8.89	0.400	25	25	25	25	0

Table B.2. Traffic Matrix Under Scenario C for 5-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix				
			1	2	3	4	5
1	11.94	0.400	0	60	30	5	5
2	12.00	0.000	30	0	10	30	30
3	8.96	0.300	50	40	0	5	5
4	9.55	0.320	15	60	5	0	20
5	9.55	0.320	15	60	5	20	0

Table B.3. Traffic Matrix Under Scenario B for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	17.00	0.000	0	15	15	15	10	5	10	10	10	5	5
2	16.42	0.880	15	0	10	10	15	15	5	5	5	10	10
3	14.18	0.760	15	10	0	10	10	5	15	15	10	5	5
4	11.94	0.640	15	10	10	0	10	10	10	10	20	2	3
5	7.46	0.400	10	15	10	10	0	10	10	10	10	8	7
6	12.69	0.680	5	15	5	10	10	0	10	10	5	15	15
7	7.46	0.400	10	5	15	10	10	10	0	20	10	5	5
8	7.46	0.400	10	5	15	10	10	10	20	0	10	5	5
9	7.46	0.400	10	5	10	20	10	5	10	10	0	10	10
10	7.46	0.400	5	10	5	2	8	15	5	5	10	0	35
11	7.46	0.400	5	10	5	3	7	15	5	5	10	35	0

Table B.4. Traffic Matrix Under Scenario C for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	12.11	0.438	0	30	25	25	1	2	5	5	5	1	1
2	13.00	0.000	25	0	5	5	25	25	3	1	1	5	5
3	11.83	0.428	25	15	0	5	1	1	25	25	1	1	1
4	10.34	0.364	30	20	10	0	2	2	2	2	30	1	1
5	8.73	0.316	15	50	1	1	0	20	1	1	1	5	5
6	12.49	0.452	5	35	1	1	5	0	1	1	1	25	25
7	9.18	0.332	20	10	35	5	1	1	0	25	1	1	1
8	9.07	0.328	20	10	35	5	1	1	25	0	1	1	1
9	7.91	0.286	20	25	20	45	2	2	2	2	0	1	1
10	9.18	0.332	5	25	1	1	10	30	1	1	1	0	25
11	9.18	0.332	5	25	1	1	10	30	1	1	1	25	0

Table 5. Traffic Matrix Under Scenario B for 15 Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix														
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16.00	0.000	0	20	20	10	10	6	6	6	6	6	2	2	2	2	2
2	15.01	0.790	30	0	15	15	10	10	3	3	2	3	3	3	2	2	2
3	15.84	0.728	20	15	0	3	3	3	15	10	10	2	2	7	6	2	2
4	8.70	0.400	20	20	3	0	10	10	3	3	3	10	10	2	2	2	2
5	12.70	0.584	5	20	3	10	0	10	3	3	3	20	15	2	2	2	2
6	8.70	0.400	5	30	3	10	15	0	3	3	3	10	10	2	2	2	2
7	8.70	0.400	10	3	30	2	2	2	0	15	10	2	2	7	7	4	4
8	13.36	0.614	3	3	15	2	2	2	10	0	5	10	3	20	15	5	5
9	8.70	0.400	20	3	30	2	2	2	15	10	0	2	2	3	3	3	3
10	8.70	0.400	2	15	3	7	30	7	3	3	2	0	20	2	2	2	2
11	8.70	0.400	2	15	3	7	30	7	3	3	2	20	0	2	2	2	2
12	11.75	0.540	3	3	8	2	2	3	2	30	3	2	2	0	10	20	10
13	11.75	0.540	3	3	8	2	2	3	2	30	3	2	2	10	0	10	20
14	8.70	0.400	3	3	5	2	2	2	5	20	3	2	2	30	10	0	10
15	8.70	0.400	3	3	5	2	2	2	5	20	3	2	2	10	30	10	0

Table 6. Traffic Matrix Under Scenario C for 15 Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix														
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	14.26	0.576	0	30	15	5	3	4	10	10	10	2	2	3	3	2	2
2	17.00	0.000	15	0	3	20	15	20	1	1	1	10	10	1	1	1	1
3	15.60	0.630	20	20	0	2	2	2	10	10	10	1	1	6	6	5	5
4	8.81	0.356	10	30	3	0	15	15	1	1	1	10	10	1	1	1	1
5	14.70	0.594	2	30	1	10	0	10	1	1	1	20	20	1	1	1	1
6	8.81	0.356	10	30	3	15	15	0	1	1	1	10	10	1	1	1	1
7	8.21	0.332	10	15	20	1	1	1	0	15	15	1	1	6	6	4	4
8	16.34	0.660	10	15	10	1	1	1	5	0	5	1	1	15	15	10	10
9	8.21	0.332	10	15	20	1	1	1	15	15	0	1	1	6	6	4	4
10	8.91	0.360	2	20	1	10	30	10	1	1	1	0	20	1	1	1	1
11	8.91	0.360	2	20	2	10	30	10	1	1	1	20	0	1	1	1	1
12	10.34	0.418	2	10	5	1	1	1	5	30	5	1	1	0	5	30	3
13	10.34	0.418	2	10	5	1	1	1	5	30	5	1	1	5	0	3	30
14	8.27	0.334	2	10	5	1	1	1	5	10	5	1	1	50	5	0	3
15	8.27	0.334	2	10	5	1	1	1	5	10	5	1	1	5	50	3	0

Table B.7. Traffic Matrix Under Scenario B for 6-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix					
			1	2	3	4	5	6
1	17.00	0.000	0	20	20	20	20	20
2	16.88	0.720	20	0	20	20	20	20
3	15.00	0.640	20	20	0	20	20	20
4	9.38	0.400	20	20	20	0	20	20
5	9.38	0.400	20	20	20	20	0	20
6	9.38	0.400	20	20	20	20	20	0

Table B.8. Traffic Matrix Under Scenario C for 6-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix					
			1	2	3	4	5	6
1	15.32	0.480	0	40	30	10	10	10
2	16.00	0.000	25	0	15	25	25	10
3	13.40	0.420	30	30	0	5	5	30
4	10.53	0.330	20	50	5	0	20	5
5	10.53	0.330	20	50	5	20	0	5
6	10.21	0.320	25	20	45	5	5	0

Table B.9. Traffic Matrix Under Scenario D for 6-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix					
			1	2	3	4	5	6
1	18.75	0.800	0	20	40	5	5	30
2	16.64	0.710	20	0	35	10	10	25
3	19.00	0.000	25	25	0	15	15	20
4	7.27	0.310	10	20	40	0	10	20
5	7.27	0.310	10	20	40	10	0	20
6	10.08	0.430	20	20	30	15	15	0

Table B.10. Traffic Matrix Under Scenario E for 6-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix					
			1	2	3	4	5	6
1	18.00	0.000	0	20	20	20	20	20
2	17.14	0.560	50	0	5	20	20	5
3	13.16	0.430	50	10	0	5	5	30
4	9.80	0.320	50	30	5	0	10	5
5	9.80	0.320	50	30	5	10	0	5
6	10.10	0.330	50	10	30	5	5	0

Table B.11. Traffic Matrix Under Scenario A for 6-Node Static LAN

Node	Arrival Rate	Service Time	Traffic Matrix					
			1	2	3	4	5	6
1	17.00	0.000	0	20	20	20	20	20
2	16.88	0.618	20	0	20	20	20	20
3	15.00	0.573	20	20	0	20	20	20
4	9.38	0.343	20	20	20	0	20	20
5	9.38	0.343	20	20	20	20	0	20
6	9.38	0.370	20	20	20	20	20	0

Table B.12. Transition Set Under 3 Reconfigurations for 6-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
15000.0	5	2
15000.0	6	3
15000.0	3	1

Table B.13. Transition Set Under 1 Reconfiguration for 6-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
30000.0	5	2

Table B.14. Traffic Matrix Under Scenario B for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	20.00	0.000	0	15	15	15	10	5	10	10	10	5	5
2	19.70	0.880	15	0	10	10	15	15	5	5	5	10	10
3	17.02	0.760	15	10	0	10	10	5	15	15	10	5	5
4	14.33	0.640	15	10	10	0	10	10	10	10	20	2	3
5	8.96	0.400	10	15	10	10	0	10	10	10	10	8	7
6	15.22	0.680	5	15	5	10	10	0	10	10	5	15	15
7	8.96	0.400	10	5	15	10	10	10	0	20	10	5	5
8	8.96	0.400	10	5	15	10	10	10	20	0	10	5	5
9	8.96	0.400	10	5	10	20	10	5	10	10	0	10	10
10	8.96	0.400	5	10	5	2	8	15	5	5	10	0	35
11	8.96	0.400	5	10	5	3	7	15	5	5	10	35	0

Table B.15. Traffic Matrix Under Scenario C for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	14.53	.438	0	30	25	25	1	2	5	5	5	1	1
2	15.00	.000	25	0	5	5	25	25	3	1	1	5	5
3	14.20	.428	25	15	0	5	1	1	25	25	1	1	1
4	12.41	.364	30	20	10	0	2	2	2	2	30	1	1
5	10.48	.316	15	50	1	1	0	20	1	1	1	5	5
6	15.00	.452	5	35	1	1	5	0	1	1	1	25	25
7	11.01	.332	20	10	35	5	1	1	0	25	1	1	1
8	10.88	.328	20	10	35	5	1	1	25	0	1	1	1
9	9.49	.286	20	25	20	45	2	2	2	2	0	1	1
10	11.01	.332	5	25	1	1	10	30	1	1	1	0	25
11	11.01	.332	5	25	1	1	10	30	1	1	1	25	0

Table B.16. Traffic Matrix Under Scenario D for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	29.02	1.336	0	10	35	10	1	1	20	20	1	1	1
2	20.07	.924	10	0	30	50	5	5	20	20	1	2	2
3	30.00	.000	20	15	0	15	5	5	15	15	5	3	2
4	12.00	.552	15	10	24	0	2	2	15	15	15	1	1
5	5.87	.270	10	15	24	5	0	5	15	15	1	5	5
6	12.51	.576	5	10	20	10	4	0	15	15	1	10	10
7	11.30	.520	25	15	20	15	3	3	0	10	3	3	3
8	11.30	.520	25	15	20	15	3	3	10	0	3	3	3
9	5.91	.272	10	5	25	15	2	1	20	20	0	1	1
10	6.04	.278	5	10	15	5	5	17	15	15	3	0	10
11	6.00	.276	5	10	15	5	5	17	15	15	3	10	0

Table B.17. Traffic Matrix Under Scenario E for 11-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	27.83	1.276	0	10	10	50	5	5	5	5	5	2	3
2	22.25	.780	10	0	10	50	5	5	2	2	10	3	3
3	16.10	.738	10	5	0	50	5	5	10	10	3	1	1
4	28.00	.000	15	15	15	0	10	10	10	10	10	3	2
5	6.94	.318	5	10	5	50	0	10	5	5	5	2	3
6	15.53	.712	5	13	5	50	10	0	1	1	5	5	5
7	6.63	.304	5	5	10	50	5	5	0	10	5	3	2
8	6.63	.304	5	5	10	50	5	5	10	0	5	2	3
9	6.81	.316	10	10	10	50	4	4	4	4	0	2	2
10	5.58	.256	5	10	5	50	5	10	3	2	5	0	5
11	5.63	.258	5	10	5	50	5	10	2	3	5	5	0

Table B.18. Traffic Matrix Under Scenario B for 11-Node Static LAN

Node	Arrival Rate	Service Time	Traffic Matrix										
			1	2	3	4	5	6	7	8	9	10	11
1	20.00	0.000	0	15	15	15	10	5	10	10	10	5	5
2	19.70	0.816	15	0	10	10	15	15	5	5	5	10	10
3	17.02	0.816	15	10	0	10	10	5	15	15	10	5	5
4	14.33	0.708	15	10	10	0	10	10	10	10	20	2	3
5	8.96	0.326	10	15	10	10	0	10	10	10	10	8	7
6	15.22	0.605	5	15	5	10	10	0	10	10	5	15	15
7	8.96	0.389	10	5	15	10	10	10	0	20	10	5	5
8	8.96	0.388	10	5	15	10	10	10	20	0	10	5	5
9	8.96	0.319	10	5	10	20	10	5	10	10	0	10	10
10	8.96	0.317	5	10	5	2	8	15	5	5	10	0	35
11	8.96	0.317	5	10	5	3	7	15	5	5	10	35	0

Table B.19. Transition Set Under 3 Reconfigurations for 11-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
6900.0	5	2
7000.0	7	3
7000.0	9	4

Table B.20. Transition Set Under 1 Reconfiguration for 11-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
13900.0	7	3

Table 21. Traffic Matrix Under Scenario B for 16-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	21.00	0.000	0	15	15	10	5	5	5	5	5	5	5	5	5	5	5	5
2	20.94	1.160	15	0	5	10	10	10	5	5	5	5	5	5	5	5	5	5
3	18.24	1.010	10	5	0	5	5	5	15	10	5	5	5	10	5	5	5	5
4	18.78	1.040	5	10	5	0	5	5	5	5	5	5	15	5	5	5	10	10
5	18.42	1.020	5	10	5	5	0	5	5	5	5	15	5	5	10	10	5	5
6	11.37	0.630	10	10	5	10	5	0	5	5	15	5	5	5	5	5	5	5
7	12.10	0.670	10	5	10	5	5	5	0	10	5	5	5	15	5	5	5	5
8	6.86	0.380	10	5	15	5	5	5	10	0	5	5	5	10	5	5	5	5
9	6.68	0.370	5	10	5	10	10	5	5	5	0	5	5	5	5	5	5	5
10	15.71	0.870	5	10	5	10	10	15	5	5	5	0	5	5	10	10	5	5
11	15.71	0.870	5	10	5	10	10	5	5	5	5	5	0	5	5	5	10	10
12	7.04	0.390	5	5	15	5	5	5	15	10	5	5	5	0	5	5	5	5
13	7.04	0.390	5	5	5	5	10	5	5	5	5	15	5	5	0	15	5	5
14	7.04	0.390	5	5	5	5	10	5	5	5	5	15	5	5	15	0	5	5
15	7.04	0.390	5	5	5	10	5	5	5	5	5	5	15	5	5	5	0	15
16	7.04	0.390	5	5	5	10	5	5	5	5	5	5	15	5	5	5	15	0

Table 22. Traffic Matrix Under Scenario C for 16-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	29.16	1.624	0	20	10	10	7	7	5	5	5	5	5	1	5	5	5	5
2	30.00	0.000	8	0	8	8	8	8	6	6	6	6	6	6	6	6	6	6
3	23.34	1.300	10	20	0	9	9	9	10	5	5	5	5	5	2	2	2	2
4	18.24	1.016	12	20	10	0	5	5	10	5	2	2	10	5	2	2	5	5
5	18.14	1.010	16	20	10	8	0	8	5	5	2	10	2	2	5	5	1	1
6	11.21	0.624	10	20	15	8	8	0	5	5	10	5	5	5	1	1	1	1
7	12.28	0.684	7	15	10	10	10	10	0	5	5	5	5	10	2	2	2	2
8	7.36	0.410	8	15	10	10	10	10	5	0	5	5	5	5	3	3	3	3
9	5.67	0.316	20	15	10	5	4	15	5	5	0	3	3	5	3	3	2	2
10	13.76	0.766	15	15	10	8	10	5	7	7	1	0	3	5	6	6	1	1
11	13.61	0.758	13	15	10	13	5	5	5	7	5	3	0	5	1	1	6	6
12	6.25	0.348	5	10	10	10	10	10	15	10	4	4	4	0	2	2	2	2
13	5.28	0.294	10	15	10	5	7	3	10	10	2	12	2	5	0	5	2	2
14	5.28	0.294	10	15	10	5	7	3	10	10	2	12	2	5	5	0	2	2
15	5.21	0.290	10	15	10	7	5	3	10	10	2	2	12	5	2	2	0	5
16	5.21	0.290	10	15	10	7	5	3	10	10	2	2	12	5	2	2	5	0

Table 23. Traffic Matrix Under Scenario D for 16-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	17.91	0.720	0	20	21	10	10	10	7	7	2	2	2	5	1	1	1	1
2	20.11	0.812	10	0	10	20	10	10	1	1	9	10	10	1	2	2	2	2
3	21.00	0.000	10	10	0	12	8	10	10	10	4	4	4	6	3	3	3	3
4	17.26	0.694	8	10	15	0	5	5	1	1	2	5	23	1	2	2	10	10
5	16.62	0.668	10	15	15	5	0	5	1	1	3	15	5	1	10	10	2	2
6	12.94	0.520	10	20	15	7	6	0	1	1	25	3	3	1	2	2	2	2
7	11.19	0.450	10	7	30	5	3	3	0	15	1	1	1	20	1	1	1	1
8	7.86	0.316	10	5	40	5	4	4	15	0	1	1	1	10	1	1	1	1
9	7.96	0.320	8	15	15	8	5	30	1	1	0	4	4	1	2	2	2	2
10	14.03	0.564	5	8	10	8	25	5	1	1	2	0	2	1	15	15	1	1
11	14.43	0.580	5	8	10	28	5	5	1	1	2	2	0	1	1	1	15	15
12	7.56	0.304	8	5	30	3	1	1	30	15	1	1	1	0	1	1	1	1
13	8.01	0.322	2	10	7	8	10	6	1	1	2	30	2	1	0	18	1	1
14	8.01	0.322	2	10	7	8	10	6	1	1	2	30	2	1	18	0	1	1
15	8.01	0.322	2	10	7	14	5	5	1	1	2	2	30	1	1	1	0	18
16	8.01	0.322	2	10	7	14	5	5	1	1	2	2	30	1	1	1	18	0

Table 24. Traffic Matrix Under Scenario E for 16-Node DHLAN

Node	Arrival Rate	Service Time	Traffic Matrix															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	19.64	0.872	0	10	11	50	4	4	5	5	1	1	3	2	1	1	1	1
2	34.51	1.532	8	0	4	50	8	8	1	1	5	5	1	1	3	3	1	1
3	16.94	0.752	9	5	0	50	1	1	10	10	1	1	1	7	1	1	1	1
4	35.00	0.000	7	8	7	0	7	7	6	6	7	7	8	4	6	6	7	7
5	16.58	0.736	4	8	2	50	0	4	1	1	2	10	1	1	7	7	1	1
6	10.05	0.446	4	10	2	50	4	0	1	1	20	2	1	1	1	1	1	1
7	10.59	0.470	7	2	15	50	2	2	0	5	1	1	1	10	1	1	1	1
8	6.58	0.292	10	5	20	50	1	1	5	0	1	1	1	1	1	1	1	1
9	6.53	0.290	4	10	3	50	3	20	1	1	0	2	1	1	1	1	1	1
10	14.24	0.632	2	3	1	50	15	2	1	1	1	0	1	1	10	10	1	1
11	11.72	0.520	2	5	2	50	2	2	1	1	1	1	0	1	1	1	15	15
12	6.04	0.268	5	1	10	50	1	1	15	10	1	1	1	0	1	1	1	1
13	6.58	0.292	1	4	1	50	10	2	1	1	1	15	1	1	0	10	1	1
14	6.58	0.292	1	4	1	50	10	2	1	1	1	15	1	1	10	0	1	1
15	6.71	0.298	2	3	1	50	1	1	1	1	1	1	20	1	1	1	0	15
16	6.71	0.298	2	3	1	50	1	1	1	1	1	1	20	1	1	1	15	0

Table 25. Traffic Matrix Under Scenario B for 16-Node Static LAN

Node	Arrival Rate	Service Time	Traffic Matrix															
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	21.00	0.000	0	15	15	10	5	5	5	5	5	5	5	5	5	5	5	5
2	20.94	1.117	15	0	5	10	10	10	5	5	5	5	5	5	5	5	5	5
3	18.24	0.946	10	5	0	5	5	5	15	10	5	5	5	10	5	5	5	5
4	18.78	1.071	5	10	5	0	5	5	5	5	5	5	15	5	5	5	10	10
5	18.42	0.859	5	10	5	5	0	5	5	5	5	15	5	5	10	10	5	5
6	11.37	0.555	10	10	5	10	5	0	5	5	15	5	5	5	5	5	5	5
7	12.10	0.569	10	5	10	5	5	5	0	10	5	5	5	15	5	5	5	5
8	6.86	0.350	10	5	15	5	5	5	10	0	5	5	5	10	5	5	5	5
9	6.68	0.324	5	10	5	10	10	5	5	5	0	5	5	5	5	5	5	5
10	15.71	0.708	5	10	5	10	10	15	5	5	5	0	5	5	10	10	5	5
11	15.71	0.682	5	10	5	10	10	5	5	5	5	5	0	5	5	5	10	10
12	7.04	0.328	5	5	15	5	5	5	15	10	5	5	5	0	5	5	5	5
13	7.04	0.325	5	5	5	5	10	5	5	5	5	15	5	5	0	15	5	5
14	7.04	0.325	5	5	5	5	10	5	5	5	5	15	5	5	15	0	5	5
15	7.04	0.325	5	5	5	10	5	5	5	5	5	5	15	5	5	5	0	15
16	7.04	0.325	5	5	5	10	5	5	5	5	5	5	15	5	5	5	15	0

Table 26. Transition Set Under 3 Reconfigurations for 16-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
4900.0	10	2
5000.0	12	3
5000.0	15	4

Table 27. Transition Set Under 1 Reconfiguration for 16-Node DHLAN

Time Left for Next Transition (milliseconds)	Initiating Node	New Apex
8900.0	12	3

Appendix C. SOURCE CODE FOR THE DHLAN SIMULATION MODEL IN SIMULA-67

```
//A246ANJL JOB 311E1,ANJALI,TIME=15,REGION=1024K,MSGLEVEL=(0,0)
/*PRIORITY IDLE
/*JOBPARM LINES=10
//STEP1 EXEC SIMCLG,PARM.SIM='NOSOURCE',REGION.LKED=512K
//SIM.SYSIN DD *

SIMULATION BEGIN

    COMMENT
        THE PROGRAM CODE FOR THE DHLAN SIMULATION MODEL WAS ORIGINALLY
    WRITTEN BY SEKKAPPA NAGAPPAN IN 1986. IT HAS BEEN REVISED BY
    ANJALI VENKATESHWARAN IN 1987. THE OBJECTIVE OF THIS PROGRAM IS TO
    COMPARE THE DYNAMIC ARCHITECTURES WITH THEIR CORRESPONDING STATIC
    ARCHITECTURES AND STUDY THE PERFORMANCE OF DYNAMIC HIERARCHICAL
    NETWORKS. THE ACTUAL CODE THAT FOLLOWS CORRESPONDS TO THE
    DYNAMIC LAN. WITH SLIGHT MODIFICATIONS THE CODE FOR THE STATIC
    LAN CAN BE OBTAINED (DELETION OF SEVERAL STATEMENTS IN THE
    CONFIGURATION CHANGER).

        THE INPUT DATA IS ORGANIZED AS FOLLOWS:

    1) NUMBER OF BATCHES FOR THE ENTIRE SIMULATION RUN ('NUM_BATCHES').
    2) NUMBER OF RECONFIGURATIONS IN THE NETWORK IN EACH BATCH
       ('NUM_CONFIG').
    3) NUMBER OF ENTRIES IN THE NETWORK CONNECTIVITY INFORMATION + 1
       (NUM_ENTRIES).
    4) NUMBER OF NODES IN THE NETWORK ('NO_NODES').
    5) NUMBER OF APEX CANDIDATES IN THE NETWORK INCLUDING THE APEX
       ('NO_APEXES').
    6) TOPOLOGY OR NETWORK CONNECTIVITY INFORMATION. THIS IS A SET OF
       ENTRIES TERMINATED BY A -1. EACH ENTRY CONSISTS OF TWO VALUES.
       THE TWO VALUES ARE NODES WHICH HAVE POINT-TO-POINT LINK. THE
       FIRST VALUE OF THE FIRST ENTRY IS THE STARTING APEX BY DEFAULT.
    7) TRANSITION INFORMATION.
       THE COLUMNS IN THE MATRIX CONSISTS OF:
            COLUMN 1: PACKET ARRIVAL RATE FOR NODES (PACKETS/MILLISECOND).
            COLUMN 2: LINK SERVICE TIME FOR NODES (MILLISECONDS).
            COLUMN 3 ONWARDS: 'NO_NODES' PACKET RATES (OUT OF 100 PACKETS
                              PER SECOND FOR THIS NODE) FOR DIFFERENT
                              DESTINATIONS.
    8) TRANSITION SET:
           COLUMN 1: TIME LEFT FOR NEXT RECONFIGURATION IN MILLISECONDS.
           COLUMN 2: INITIATING NODE.
           COLUMN 3: NEW APEX.
    THE NUMBER OF TRANSITION SETS IS EQUAL TO THE NUMBER OF RECONFIGURAT-
    IONS AND THE NUMBER OF TRAFFIC MATRICES IS EQUAL TO (1 + NUMBER OF
    RECONFIGURATIONS).

    HARD-CODED PARAMETERS:

    1) OBSERVATIONS FOR STEADY STATE (45000).
    2) OBSERVATIONS IN STEADY STATE (BATCH SIZE = 30000).
    3) SLEEP TIME (ECHO SUM OF ACS NODES QUEUE LENGTHS)
       (500.0 MILLISECONDS).
    4) TRANSITION TIME (RECONFIGURATION LENGTH) (300.0 MILLISECONDS).

    OUTPUT:
    1) ECHOING OF SOME INPUT PARAMETERS.
```

```
     2) TIME AT WHICH THE SPECIFIED NUMBER OF PACKETS ARE DELIVERED
        FOR STEADY STATE CONDITIONS.
     3) QUEUE LENGTHS AT EACH NODE AT START OF DATA COLLECTION.

     FOR EACH BATCH:

     1) TOTAL DATA QUEUE LENGTH OF ALL APEX CANDIDATES AT REGULAR INTERVAL
        OF TIME 500.0 MILLISECONDS.
     2) QUEUE LENGTHS AT EACH NODE AT END (OF BATCH).
     3) NUMBER OF TRANSITIONS OCCURED IN THE NETWORK.
     4) NUMBER OF PACKETS GENERATED.
     5) NUMBER OF PACKETS RECEIVED.
     6) SIMULATION TIME
     7) TOTAL DELAY
     8) AVERAGE PACKET TRANSMISSION DELAY.
     9) NETWORK THROUGHPUT.
    10) NETWORK DELIVERY RATIO.

     AFTER SPECIFIED NUMBER OF BATCHES (5, 10, 15, 20, 40, ...) THE
     FOLLOWING INFORMATION IS PRINTED:

     1. BATCH AVERAGE DELAYS, THROUGHPUT, NETWORK DELIVERY RATIOS.
     2. OVERALL MEAN OF THE ABOVE NETWORK PARAMTERS.
     3. AUTOCOVARIANCE FUNTION FOR THE THREE PARAMETERS.
     4. AUTOCORRELATION BETWEEN THE BATCH MEANS FOR THE THREE PARAMETERS.
     5. VARIANCE BETWEEN THE BATCH MEANS FOR EACH PARAMETER.
     6. 95% CONFIDENCE INTERVAL AND HALF-WIDTH FOR THE THREE PARAMETERS.
        ;


COMMENT THIS PROCEDURE(FUNCTION) IS USED TO COMPARE TWO INTEGERS AND PICK
        THE LARGER ONE;
INTEGER PROCEDURE MAX (A,B);
   INTEGER A,B;

   BEGIN
      COMMENT NO EXPLANATION IS REQUIRED FOR THE FOLLOWING CODE;
      IF (A>B) THEN MAX := A
      ELSE MAX := B;
   END;



COMMENT THIS PROCEDURE(FUNCTION) IS USED TO COMPARE TWO REALS AND PICK
        THE SMALLER ONE;
REAL PROCEDURE MINR (A,B);
   REAL A,B;

   BEGIN
      COMMENT NO EXPLANATION IS REQUIRED FOR THE FOLLOWING CODE;
      IF (A<B) THEN MINR := A
      ELSE MINR := B;
   END OF MINR;



COMMENT THIS PROCEDURE(FUNCTION) IS USED TO COMPARE TWO REALS AND PICK
        THE LARGER ONE;
REAL PROCEDURE MAXR (A,B);
   REAL A,B;

   BEGIN
      COMMENT NO EXPLANATION IS REQUIRED FOR THE FOLLOWING CODE;
      IF (A>B) THEN MAXR := A
      ELSE MAXR := B;
   END OF MAXR;
```

```
COMMENT THIS IS USED TO PRINT A LINE OF STRING(TEXT) FOLLOWED BY A
        CARRIAGE RETURN OR NEW LINE;
PROCEDURE OUTLINE(T);
   VALUE T;
   TEXT  T; COMMENT POINTER TO THE TEXT TO BE PRINTED;
   BEGIN
      OUTTEXT(T);       COMMENT OUTPUT THE TEXT;
      OUTIMAGE;         COMMENT PRINT A CARRIAGE RETURN OR NEWLINE;
   END OF OUTLINE;


COMMENT THIS PROCEDURE IS USED FOR PRINTING SOME TEXT, FOLLOWED BY A REAL
        AND A NEW LINE;
PROCEDURE OUTR(T,R);
   VALUE T; TEXT T;     COMMENT TEXT TO PRECEDE THE REAL NUMBER;
   REAL R;              COMMENT THE REAL NUMBER TO BE PRINTED;
   BEGIN
      OUTTEXT(T);       COMMENT PRINT THE TEXT;
      OUTFIX(R,3,10);   COMMENT PRINT THE REAL NUMBER;
      OUTIMAGE;         COMMENT PRINT A CARRIAGE RETURN OR NEWLINE;
   END OF OUTR;



COMMENT THIS PROCEDURE IS USED FOR PRINTING SOME TEXT, FOLLOWED BY AN
        INTEGER AND A NEW LINE;
PROCEDURE OUTI(T,I);
   VALUE T; TEXT T;  COMMENT TEXT TO PRECEDE THE INTEGER;
   INTEGER I;        COMMENT THE INTEGER TO BE PRINTED;
   BEGIN
      OUTTEXT(T);    COMMENT PRINT THE TEXT;
      OUTINT(I,8);   COMMENT PRINT THE INTEGER;
      OUTIMAGE;      COMMENT PRINT A CARRIAGE RETURN OR NEWLINE;
   END OF OUTI;




COMMENT THIS IS A CLASS DEFINITION FOR THE ABSTRACT DATA TYPE PACKET.
        THE PACKET DOES NOT CONTAIN ALL FIELDS IN THE PROTOCOL FOR THE
        PURPOSES OF THIS MODEL. ONE PRIMARY REASON IS THAT THEY ARE
        NOT REQUIRED. SECONDLY IT SAVED ENORMOUS AMOUNT OF MEMORY SPACE.
        SO IT CONTAINS THE ESSENTIAL FIELDS OF:
           SOURCE NODE OF THE PACKET,
           DESTINATION NODE OF THE PACKET,
           TYPE OF PACKET (DATA, ACK, REJECT ETC),
           DATA (CONSIDERED INTEGER, NOT A STRING OF BYTES)
        AND SOME BOOKKEEPING INFORMATION, SUCH AS:
           THE TIME AT WHICH THE PACKET WAS ORIGINATED
           THE SEQUENCE NUMBER OF THE PACKET
        AS PACKETS ARE GOING TO QUEUED MANY TIMES DURING ITS LIFETIME,
        AND AS THE QUEUES ARE ESSENTIALLY FORMED BY CHAINING PACKETS
        IN A PARTICULAR QUEUE WE REQUIRE A FIELD TO CONTAIN A POINTER
        TO A PACKET;

CLASS PACKET (SRC,DEST,TYPE);
   INTEGER SRC;   COMMENT SOURCE NODE OF THE PACKET;
   INTEGER DEST;  COMMENT DESTINATION NODE OF THE PACKET;
   INTEGER TYPE;  COMMENT TYPE OF THE PACKET;

   BEGIN
      REAL START_TIME;     COMMENT TIME THIS PACKET WAS ORIGINATED;
      INTEGER DATA;        COMMENT THIS IS THE DATA FIELD;
      REF (PACKET) NEXT;   COMMENT POINTER TO NEXT PACKET IN QUEUE,IF ANY;
```

```
      COMMENT UPDATE THE COUNT OF PACKETS ORIGINATED BY A PARTICULAR
            NODE;
      NET.NODE(SRC).NO_PKTS := NET.NODE(SRC).NO_PKTS+1;

      COMMENT UPDATE THE GLOBAL VARIABLE CONTAINING
      THE SEQUENCE NUMBER FOR THE NEXT PACKET;
      SEQNR := SEQNR + 1;
      COMMENT OUTTEXT("SEQ "); COMMENT OUTINT(SEQNR,3);
   END OF PACKET;



COMMENT THIS PROCEDURE IS USED FOR POSTPROCESSING A PACKET (SOME
        BOOK-KEEPING AND STATISTICS COLLECTION PROCEDURES WHEN A PACKET
        IS DELIVERED AT ITS DESTINATION);

PROCEDURE RCD(PKT);
   REF (PACKET) PKT;  COMMENT POINTER TO THE PACKET RECEIVED (THIS IS
                              ALSO THE PACKET TO BE DESTROYED);
   BEGIN
        COMMENT UPDATE THE GLOBAL VARIABLE (SYSTEM VARIABLE) CONTAINING
                THE NUMBER OF PACKETS DELIVERED SO FAR IN THE SYSTEM.
                THIS IS USED IN THROUGHPUT CALCULATIONS;
        RECEIVED := RECEIVED + 1;

        COMMENT CHECK FOR STEADY STATE, IF NOT THEN IF NUMBER OF PKTS
        RECEIVED IS EQUAL TO THE NUMBER REQUIRED FOR STEADY STATE THEN
        REINITIALIZE THE STATISTICS COLLECTORS;

        IF NOT(STEADY_STATE) THEN BEGIN
           IF RECEIVED=OBS_FOR_STEADY_STATE THEN
              NET.REINITIALIZE_DELETE_STATISTICS;
           END
        ELSE BEGIN

             COMMENT NOTE THE PACKET TRANSMISSION DELAY, TOTAL DELAY,
             AND THE AVERAGE TRANSMISSION DELAY FOR THE NUMBER OF PKTS
             RECEIVED TILL THAT POINT. DO THIS TILL
             THE NUMBER OF PACKETS RECEIVED EQUALS THE NUMBER OF
             OBSERVATIONS IN STEADY STATE;

             COMMENT MAKE ALL REFERENCE WITHIN THIS PACKET;
             INSPECT PKT DO
               BEGIN
                 IF NOT(RECEIVED>OBS_IN_STEADY_STATE) THEN BEGIN

                    COMMENT UPDATE THE GLOBAL VARIABLE (SYSTEM VARIABLE)
                    CONTAINING SUM OF DELAYS EXPERIENCED BY ALL PACKETS
                    DELIVERED SO FAR. THIS IS USED IN TRANSMISSION-DELAY
                    CALCULATIONS;
                    TOTAL_DELAY := TOTAL_DELAY + TIME - PKT.START_TIME;

                    COMMENT PROBES FOR VERIFICATION PURPOSES;
                    COMMENT OUTINT(RECEIVED,6)
                    OUTFIX(PKT.START_TIME,6,18)
                    OUTFIX(TIME,6,18)
                    OUTFIX(TOTAL_DELAY,2,10)
                    OUTIMAGE;
                    END;
                END;
          IF RECEIVED=OBS_IN_STEADY_STATE THEN ACTIVATE NET.NEW_BATCH;
         END;
   END OF RCD;
```

```
COMMENT THIS IS A STRUCTURE (CLASS) USED FOR ALL QUEUEING MECHANISMS
        WITHIN THE MODEL. THIS WAS DESIGNED (INSTEAD OF CHOOSING THE
        SIMULA 'HEAD') BECAUSE OF CERTAIN CUSTOM REQUIREMENT FOR
        STATISTICS COLLECTION.
        THE QUEUING DISCIPLINE IS FCFS (FIFO).
        IT MAINTAINS FOR EACH QUEUE THE FOLLOWING INFORMATION:
           PACKETS CURRENTLY IN THE QUEUE
           TOTAL PACKETS EVER VISITED THIS QUEUE
           SUM OF TIME SPENT BY ALL PACKETS EVER VISITED THIS QUEUE
           THE LAST TIME THE QUEUE WAS MODIFIED (CHANGED BY ENTRY OR
               EXIT OF A PACKET)
           POINTER TO THE FIRST PACKET IN THE QUEUE
           POINTER TO THE LAST PACKET IN THE QUEUE
           MAXIMUM LENGTH THIS QUEUE EVER REACHED
        GLOBAL OPERATIONS WHICH ARE ALLOWED ON A QUEUE ARE:
           APPENDING A PACKET TO THE QUEUE
           REMOVING THE FIRST PACKET FROM THE QUEUE
        AND THE A PARTICULARE QUEUE CAN BE CHECKED IF ITS EMPTY OR NOT;

CLASS QUEUE;
   BEGIN

      COMMENT THE LAST TIME THE QUEUE WAS MODIFIED (CHANGED BY ENTRY OR
              EXIT OF A PACKET);
      REAL LAST_TIME;

      COMMENT  SUM OF TIME SPENT BY ALL PACKETS EVER VISITED THIS QUEUE;
      REAL TIME_X_N;

      INTEGER CARDINAL;  COMMENT PACKETS CURRENTLY IN THE QUEUE;
      INTEGER MAXLEN;    COMMENT MAXIMUM LENGTH THIS QUEUE EVER REACHED;
      INTEGER TOTAL;     COMMENT TOTAL PACKETS EVER VISITED THIS QUEUE;

      COMMENT POINTER TO THE FIRST PACKET IN THE QUEUE;
      REF (PACKET) FIRST;

      COMMENT POINTER TO THE FIRST PACKET IN THE QUEUE;
      REF (PACKET) LAST;

      COMMENT THIS PROCEDURE IS USED TO CHECK IF A QUEUE IS EMPTY. IF
              THE QUEUE IS EMPTY IT RETURNS 'TRUE';
      BOOLEAN PROCEDURE EMPTY;
         BEGIN
            COMMENT IF THE NUMBER OF PACKETS IN THE QUEUE IS ZERO, THEN
                    THE QUEUE IS EMPTY;
            EMPTY := CARDINAL = 0;
         END OF EMPTY;

      COMMENT THIS PROCEDURE IS USED TO APPEND A PACKET ( 'X' POINTER
              TO THE PACKET TO BE APPENDED) TO A QUEUE;
      PROCEDURE ADDQ(X);
         REF (PACKET) X;   COMMENT PACKET TO BE APPENDED;
         BEGIN
            X.NEXT :- NONE;   COMMENT MAKE THIS PACKET THE LAST PACKET;
            TOTAL := TOTAL +1; COMMENT UPDATE THE NO. OF PACKETS;

            COMMENT CHECK IF THE QUEUE IS EMPTY;
            IF ( LAST == NONE ) THEN
               BEGIN
                  COMMENT MAKE THIS PACKET THE FIRST AND THE LAST PACKET
                          OF THE QUEUE, AS THE QUEUE WAS EMPTY BEFORE;
                  LAST :- X;
                  FIRST :-X
               END
```

```
            ELSE BEGIN
                    COMMENT LINK THIS PACKET TO THE LAST PACKET;
                    LAST.NEXT :- X;

                    COMMENT MAKE THIS PACKET THE LAST PACKET;
                    LAST :- X
                 END;

            COMMENT UPDATE THE TIME PRODUCT NO. OF PACKETS IN THE QUEUE.
                    THIS IS USED FOR CALCULATING AVERAGE TIME SPENT
                    BY A PACKET IN THIS QUEUE;
            TIME_X_N := TIME_X_N + (TIME - LAST_TIME)*CARDINAL;

            COMMENT INCREMENT THE QUEUE COUNT;
            CARDINAL := CARDINAL + 1;

            COMMENT IF THIS COUNT IS THE MAXIMUM REACHED SO FAR, UPDATE
                    MAXIMUM QUEUE LENGTH EVER REACHED;
            MAXLEN   := MAX(MAXLEN,CARDINAL);

            COMMENT RECORD THE TIME OF CHANGE IN THE QUEUE;
            LAST_TIME := TIME;

         END OF ADDQ;


      COMMENT THIS PROCEDURE REMOVES THE FIRST PACKET FROM THE QUEUE
      AND UPDATES THIS QUEUE INFORMATION;

      REF (PACKET) PROCEDURE REMOVEQ;
         BEGIN
            IF (CARDINAL > 0) THEN
            BEGIN
               TIME_X_N := TIME_X_N + (TIME - LAST_TIME)*CARDINAL;
               CARDINAL := CARDINAL - 1;
               LAST_TIME := TIME;
               REMOVEQ :- FIRST;
               IF CARDINAL = 0 THEN
               BEGIN
                  FIRST :- NONE;
                  LAST  :- NONE;
               END
               ELSE FIRST :- FIRST.NEXT;
            END
            ELSE REMOVEQ :- NONE;
         END;


   END;



COMMENT PROCEDURE LOWER RETURNS TRUE IF THE PACKET DESTINATION NODE IS
LOWER IN HIERARCHY TO THE CURRENT NODE AT WHICH THE PACKET RESIDES,
(THE DESTINATION IS REACHABLE BY TRAVERSING THE SUCCESSORS). THE
ROUTING TABLE CONTAINS A NON ZERO VALUE IF THE DESTINATION CAN BE
REACHED THROUGH A SUCCESSOR;

BOOLEAN PROCEDURE LOWER (MY,ADDR);
   REF (NODE_CLASS) MY;
   INTEGER ADDR;

   BEGIN
         LOWER := MY.SUCC(ADDR) > 0;
   END;
```

```
COMMENT THIS IS A CLASS DEFINITION FOR THE PACKET RECEIVING FUNCTIONS
OF A NODE. IT IS SAME FOR ALL TYPES OF NODES (APEX, APEX CANDIDATE,
ORDINARY). DIFFERENT CLASSES OF NODES DIFFER PRIMARILY IN THE CODE
OR FLOW OF CONTROL. THEY HAVE THE SAME ATTRIBUTES. ONE OF THE
ATTRIBUTES IDENTIFIES THEIR CLASS AND THUS THEY TAKE THE APPROPRIATE
ACTION. MOST OF THE PROTOCOL DRAMA IS ENACTED HERE.

EACH NODE HAS ONE RECEIVE PROCESS, WHOSE MAIN FUNCTION IS TO RETRIEVE
PACKETS FROM THE NODAL INPUT QUEUES (DATA/CONTROL), INTERPRET, PROCESS
AND FILE THEM IN THE RESPECTIVE OUTPUT QUEUES OF THEIR NODES.

THE PARAMETER 'MY' INDICATES WHICH NODE THEY ARE SERVING.

THE RECEIVE PROCESS IS ACTIVE AS LONG AS THERE ARE PACKETS IN THE INPUT
QUEUES OF THE NODE IT IS SERVING. IT DEACTIVATES ITSELF
WHEN THE QUEUE BECOMES EMPTY. IT NEED TO BE ACTIVATED WHEN A NEW
PACKET IS FILED IN ONE OF THE INPUT QUEUES. THE PACKETS IN THE
CONTROL QUEUE HAVE A HIGHER PRIORITY THAN THOSE IN THE DATA QUEUES;

PROCESS CLASS R_PROCESS (MY);
   REF (NODE_CLASS) MY; COMMENT THIS IS THE NODE THIS PROCESS IS
                              SERVING;
   BEGIN

      COMMENT CODE TO BE EXECUTED IF A PARTICULAR NODE IS AN APEX;

      PROCEDURE DO_APEX_JOB;
      BEGIN


         INTEGER I;
         COMMENT VARIABLES USED DURING APEX TRANSITION;
         INTEGER NEWAPEX; COMMENT INDICATES THE INCUMBENT APEX;
         INTEGER ACK_COUNT; COMMENT NUMBER OF APEX CANDIDATES THAT
                            HAVE SENT A TRANSITION ACKNOWLEDGE TILL NOW;


         COMMENT PROCEDURE USED WHENEVER THE APEX WANTS TO BROADCAST A
         SAME TYPE OF MESSAGE TO ALL APEX CANDIDATES. THE PARAMETER
         PASSED INDICATES THE TYPE OF THE MESSAGE TO BE SENT TO THE
         APEX CANDIDATES (VOTE, CHANGE, OVER);

         PROCEDURE BROADCAST(TYPE);
         INTEGER TYPE; COMMENT INTEGER TYPE;
         BEGIN
            REF (PACKET) PKT; COMMENT POINTER TO THE PACKET TO BE SENT;
            INTEGER I;

            COMMENT SEND TO ALL APEX CANDIDATES, EXCEPT THE APEX NODE.
            NO_APEXES = APEX + NUMBER OF ACS NODES;

            FOR I:=1 STEP 1 UNTIL NO_APEXES DO
               BEGIN
                  IF NOT I=MY.ADDR THEN
                  BEGIN
                     PKT :- NEW PACKET (MY.ADDR,I,TYPE);
                     PKT.START_TIME:=TIME;
                     COMMENT OUTTEXT("  STR  ")
                     OUTFIX(PKT.START_TIME,4,8) OUTIMAGE;

                     COMMENT ADD TO THE APPROPRIATE TRANSMISSION
                     QUEUE AND ACTIVATE THE CORRESPONDING TRANSMISSION
                     PROCESS;

                     IF (LOWER(MY,I)) THEN
```

```
                        BEGIN

COMMENT
THIS WOULD SPEW OUT ALL Q STATISTICS FOR THIS NODE'S OUT_LCQ. SIMILAR
STATEMENTS COULD BE USED IN OTHER PLACES FOR TRACING PURPOSES;
OUTTEXT("NODE : "); OUTINT(MY.ADDR,2); PRINTQ(MY.OUT_LCQ);
                           MY.OUT_LCQ.ADDQ(PKT);
                           ACTIVATE MY.TL_PROCESS;
                        END
                     ELSE  BEGIN
                           MY.OUT_UCQ.ADDQ(PKT);
                           ACTIVATE MY.TU_PROCESS;
                        END;
                  END;
               END;
            END;


COMMENT DO THE FOLLOWING FUNCTIONS AS LONG AS I REMAIN AN APEX, AND
        MY CLASS DOES NOT CHANGE TO AN APEX CANDIDATE (OCCURS AT THE
        END OF APEX TRANSITION);

            WHILE (MY.ISAPEX) DO
               BEGIN
               REF (PACKET) PKT;

            COMMENT PROCESS ALL CONTROL MESSAGES BEFORE LOOKING AT THE
                    DATA QUEUE, SINCE THE CONTROL PACKETS HAVE A HIGHER
                    PRIORITY OVER THE DATA PACKETS. AT THE SAME TIME
                    CHECK TO SEE IF MY STATUS HAS NOT CHANGED TO AN APEX
                    CANDIDATE NODE;

                  IF (MY.IN_MQ.EMPTY AND MY.IN_CQ.EMPTY) THEN PASSIVATE;
                  WHILE ((NOT MY.IN_CQ.EMPTY) AND MY.ISAPEX) DO
                     BEGIN
                        PKT :- MY.IN_CQ.REMOVEQ;

                  COMMENT SIMULATE THE TRANSMISSION TIME REQUIRED FOR THIS
                          PACKET, ONLY IF THE PACKET IS NOT ADDRESSED FOR
                          ME. ALL NODAL SERVICE TIMES REFLECT THE TRANSMIS-
                          SION DELAY ONLY. A LINK'S TRANSMISSION DELAY IS
                          SIMULATED BY ONE OF ITS NODE WHICH IS LOWER IN
                          THE HIERARCHY;

                  COMMENT COLLECT STATISTICS (PACKET DELAY TO REACH ITS
                          DESTINATION) SINCE PACKET IS FOR ME;
                        IF(PKT.DEST=MY.ADDR) THEN
                        BEGIN
                           RCD(PKT);
COMMENT TRANSITION PROTOCOL DRAMA IS ENACTED HERE.
IF PACKET TYPE IS TRANSITION REQUEST THEN SET THE COUNTER FOR
RECEIVING TA'S TO THE NUMBER OF APEX CANDIDATES EXCLUDING THE CURRENT
APEX. NOTE DOWN THE INCUMBENT APEX. SEND TRANSITION VOTE TO ALL ACS
NODES EXCLUDING THE CURRENT APEX;
                           IF(PKT.TYPE=REQUEST) THEN
                           BEGIN
                              ACK_COUNT := NO_APEXES-1;
                              NEWAPEX := PKT.DATA;
                              BROADCAST(VOTE);
                           END

COMMENT TYPE IS TRANSITION ACKNOWLEDGE THEN DECREMENT ACK. COUNTER BY 1.
IF COUNTER EQUAL 0, IT IMPLIES THAT ALL ACS NODES SENT 'YES' FOR
TRANSITION. SEND TRANSITION CHANGE TO ALL ACS NODES. WAIT FOR TRANSITION
TIME. BUILD THE NEW ROUTING TABLES. READ NEW SERVICE TIMES. BROADCAST
```

```
TRANSITION OVER TO ACS NODES. AT THIS POINT THE NEWAPEX ASSUMES THE
ROLE OF LOGICAL HEAD OF THE HIERARCHY;
                        ELSE IF(PKT.TYPE=ACK) THEN
                        BEGIN
                           ACK_COUNT := ACK_COUNT -1;
                           IF (ACK_COUNT =0) THEN
                           BEGIN
                              TRANSITION := TRUE;
                              BROADCAST(CHANGE);
                              HOLD(TRANSIT_TIME);
                              NET.ZEROING;
                              NET.BUILD(NEWAPEX,0);
                              COMMENT NET.PRINTROUTINE;
                              NET.READ_SER_PRNT;
                              BROADCAST(OVER);
                              TRANSITION := FALSE;
                              MY.ISAPEX := FALSE;
                              NET.NODE(NEWAPEX).ISAPEX := TRUE;
                              APEX := NEWAPEX;
                              COMMENT NET.CHECK;
                           END;
                          END;
                        END

COMMENT PACKET DESTINATION NOT ME. HENCE SERVICE THE PACKET AND FILE THE
PACKET IN THE LOWER CONTROL QUEUE;
                    ELSE BEGIN
            HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
                         MY.OUT_LCQ.ADDQ(PKT);
                         ACTIVATE MY.TL_PROCESS;
                         END
                  END;

COMMENT THIS SECTION OF THE CODE COMES INTO ACTION ONLY WHEN A TRANSITION
TAKES PLACE AND A NEW APEX TAKES OVER. SINCE THE NEW APEX NOW RESIDES AT
HIGHEST LEVEL OF HIERARCHY, THE PACKETS EARLIER QUEUED IN THE UPPER
MESSAGE QUEUE HAVE TO BE REMOVED AND QUEUED TO LOWER MESSAGE QUEUE.
BECAUSE ALL NODES THAT WERE HIGHER IN HIERARCHY TO THIS NODE (NEWAPEX)
ARE NOW LOWER IN LEVEL;

          WHILE ((NOT MY.OUT_UMQ.EMPTY) AND MY.ISAPEX) DO BEGIN
                  PKT :- MY.OUT_UMQ.REMOVEQ;
                  MY.OUT_LMQ.ADDQ(PKT);
                  END;

                IF ((NOT MY.IN_MQ.EMPTY) AND MY.ISAPEX) THEN
                   BEGIN
                      PKT :- MY.IN_MQ.REMOVEQ;
                    IF NOT PKT.DEST = MY.ADDR THEN
                       BEGIN
           HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
                         MY.OUT_LMQ.ADDQ(PKT);
                         ACTIVATE MY.TL_PROCESS;
                       END
                    ELSE BEGIN
                         RCD(PKT);

                    END;
                  END;
             END;
       END;

COMMENT DO THE FOLLOWING AS LONG AS I AM AN APEX CANDIDATE NODE AND NOT
AN APEX NODE;
       PROCEDURE DO_APEXC_JOB;
       BEGIN
          WHILE (NOT MY.ISAPEX) DO
```

```
        BEGIN
        REF (PACKET) PKT;
           IF (MY.IN_MQ.EMPTY AND MY.IN_CQ.EMPTY) THEN PASSIVATE;
           WHILE ((NOT MY.IN_CQ.EMPTY) AND (NOT MY.ISAPEX)) DO
              BEGIN
                 PKT :- MY.IN_CQ.REMOVEQ;

COMMENT IF PACKET IS FOR ME THEN COLLECT STATISTICS;
                 IF(PKT.DEST = MY.ADDR) THEN
                 BEGIN
                    RCD(PKT);

COMMENT CHECK TYPE OF THE CONTROL PACKET. IF TYPE IS TRANSITION VOTE,
THEN SEND ACKNOLEDGE FOR TRANSITION TO APEX ('YES'). ALL APEX NODES
SEND YES TO VOTE IN THIS MODEL;
                    IF(PKT.TYPE=VOTE) THEN
                       BEGIN
                          PKT :- NEW PACKET(MY.ADDR,APEX,ACK);
                          PKT.START_TIME:=TIME;
                          COMMENT OUTTEXT("  STR  ")
                          OUTFIX(PKT.START_TIME,4,8) OUTIMAGE;
                          MY.OUT_UCQ.ADDQ(PKT);
                          ACTIVATE MY.TU_PROCESS;
                       END
                    ELSE BEGIN

COMMENT IF TYPE IS TRANSITION CHANGE THEN START BUFFERING ALL MESSAGES
HEADED UPWARDS IN THE HIERARCHY;
                       IF(PKT.TYPE=CHANGE ) THEN
                          BEGIN
                             CANCEL(MY.TU_PROCESS);
                          END

COMMENT IF TRANSITION IS OVER THEN START UNBUFFERING MESSAGES AND
ACTIVATE THE UPPER TRANSMIT PROCESS TO RESUME NORMAL OPERATION;
                       ELSE IF(PKT.TYPE=OVER) THEN
                                ACTIVATE MY.TU_PROCESS;
                       END;
                    END
                 ELSE
                    BEGIN

                     IF LOWER(MY,PKT.DEST) THEN
                        BEGIN
           HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
                           MY.OUT_LCQ.ADDQ(PKT);
                           ACTIVATE MY.TL_PROCESS;
                        END
                     ELSE BEGIN
                        HOLD(NEGEXP(MY.SERVICE_TIME,U2)) ;
                        MY.OUT_UCQ.ADDQ(PKT);
                        IF(NOT TRANSITION) THEN ACTIVATE MY.TU_PROCESS;
                     END;
                    END;
              END;

              IF ((NOT MY.IN_MQ.EMPTY) AND (NOT MY.ISAPEX)) THEN
                 BEGIN
                    PKT :- MY.IN_MQ.REMOVEQ;
                    IF NOT PKT.DEST = MY.ADDR THEN
                      BEGIN
                       IF LOWER(MY,PKT.DEST) THEN
                       BEGIN
           HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
                          MY.OUT_LMQ.ADDQ(PKT);
                          ACTIVATE MY.TL_PROCESS;
                       END
```

```
                    ELSE BEGIN
                       HOLD(NEGEXP(MY.SERVICE_TIME,U2)) ;
                       MY.OUT_UMQ.ADDQ(PKT);
                       IF(NOT TRANSITION) THEN ACTIVATE MY.TU_PROCESS;
                    END
                  END
                ELSE BEGIN
                   RCD(PKT);
                END;
             END;
      END ;
   END;


COMMENT DO THE FOLLOWING IF I AM AN ORDINARY NODE. ORDINARY NODES HAVE
NO ROLE IN TRANSITION ENACTION;
   PROCEDURE DO_NODE_JOB;
   BEGIN
        WHILE (TRUE) DO
           BEGIN
           REF (PACKET) PKT;
              IF (MY.IN_MQ.EMPTY AND MY.IN_CQ.EMPTY) THEN PASSIVATE;
              WHILE (NOT MY.IN_CQ.EMPTY) DO

COMMENT IF CONTROL QUEUE IS NOT EMPTY THEN
REMOVE PACKET FROM CONTROL QUEUE.
IF PACKET IS DESTINED TO A LOWER NODE, THEN SERVICE THE PACKET.
IT SHOULD BE NOTED THAT THE SUCCESSOR NODE INCORPORATES THE SERVICE TIME
FOR THE LINK CONNECTING THIS NODE AND THE SUCCESSOR NODE. IF PACKET IS
DESTINED TO A HIGHER NODE THEN SERVICE THE PACKET AND FILE IT IN THE
UPPER QUEUE. IF PACKET FOR ME THEN COLLECT STATISTICS;
                    BEGIN
                       PKT :- MY.IN_CQ.REMOVEQ;
                       IF(NOT PKT.DEST=MY.ADDR) THEN
                       BEGIN
                          IF LOWER(MY,PKT.DEST) THEN
                          BEGIN
         HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
                             MY.OUT_LCQ.ADDQ(PKT);
                             ACTIVATE MY.TL_PROCESS;
                          END
                          ELSE BEGIN
                             HOLD(NEGEXP(MY.SERVICE_TIME,U2));
                             MY.OUT_UCQ.ADDQ(PKT);
                             ACTIVATE MY.TU_PROCESS;
                          END;
                       END
                       ELSE BEGIN
                          RCD(PKT);
                       END;
                    END;

COMMENT THE MESSAGE QUEUE IS EXAMINED ONLY IF THE CONTROL QUEUE IS EMPTY.
SINCE CONTROL PACKETS HAVE HIGHER PRIORITY THAN MESSAGE PACKETS. REMOVE
THE PACKET FROM THE MESSAGE QUEUE. IF PACKET DESTINATION IS NOT ME THEN
SERVICE THE PACKET AND FILE IT IN THE APPROPRIATE QUEUE AND ACTIVATE THE
TRANSMIT PROCESS. IN SIMULA IF A PROCESS IS ALREADY ACTIVE THEN AN
ACTIVATE STATEMENT HAS NO EFFECT. IF PACKET IS FOR ME THEN COLLECT
STATISTICS AND REMOVE PACKET FROM THE SYSTEM;
                 IF (NOT MY.IN_MQ.EMPTY) THEN
                    BEGIN
                       PKT :- MY.IN_MQ.REMOVEQ;
                       IF NOT PKT.DEST = MY.ADDR THEN
                         BEGIN
                          IF LOWER(MY,PKT.DEST) THEN
                          BEGIN
         HOLD(NEGEXP(NET.NODE(MY.SUCC(PKT.DEST)).SERVICE_TIME,U2));
```

```
                              MY.OUT_LMQ.ADDQ(PKT);
                              ACTIVATE MY.TL_PROCESS;
                           END
                           ELSE BEGIN
                              HOLD(NEGEXP(MY.SERVICE_TIME,U2));

COMMENT THE PACKET IS DESTINED FOR A UPPER NODE HENCE FILE IT IN THE
TRANSMIT UPPER MESSAGE QUEUE AND ACTIVATE THE UPPER TRANSMIT PROCESS;
                              MY.OUT_UMQ.ADDQ(PKT);
                              ACTIVATE MY.TU_PROCESS;
                           END
                         END
                      ELSE BEGIN
                         RCD(PKT);
                      END;
                   END;
         END;
      END;

      WHILE (TRUE) DO
      BEGIN
         INSPECT MY DO
         IF (ISAPEX) THEN DO_APEX_JOB
         ELSE IF (ISAPEXCAN) THEN
                 BEGIN
                    DO_APEXC_JOB;
                    ACTIVATE NET.CHANGER;
                 END
              ELSE DO_NODE_JOB;
      END;
END;



COMMENT
THIS IS A CLASS DEFINITION FOR THE TRANSMIT FUNCTIONS OF A NODE.
EACH NODE (APEX, APEX CANDIDATE, AND ORDINARY NODE) HAS TWO SUCH
PROCESSES, SERVING TWO DIFFERENT TRANSMIT QUEUES. THE TWO SET (CONSISTING
OF CONTROL AND DATA QUEUES) CORRESPOND TO PACKETS DESTINED FOR LOWER
AND HIGHER NODES IN THE HIERARCHY WITH RESPECT TO THE NODE THESE TRANSMIT
PROCESSES ARE SERVING.

THEIR FUNCTION IS TO REMOVE PACKETS FROM THEIR TRANSMIT QUEUE AND FILE IN
INPUT QUEUES OF THE NEXT NODE IN THE PATH (DEPENDING ON WHETHER IT IS
HIGHER OR LOWER NODE).

THE PARAMETERS PASSED ARE "MY" (THE NODE WHICH THE TRANSMIT PROCESS IS
SERVING), "OUT_CQ" (THE CONTROL QUEUE WHICH THE PROCESS IS SERVING FOR
THIS NODE) AND THE "OUT_MQ" (THE DATA QUEUE WHICH THE PROCESS IS SERVING
FOR THIS NODE);

PROCESS CLASS T_PROCESS(MY,OUT_CQ,OUT_MQ);

COMMENT
THE NODE, THE CONTROL QUEUE AND THE DATA QUEUE THIS PROCESS IS SERVING;
   REF (NODE_CLASS) MY;
   REF (QUEUE) OUT_CQ,OUT_MQ;


   BEGIN
      REF(PACKET) PKT;
      REF (NODE_CLASS) NEXT_NODE;

      WHILE (TRUE) DO
         BEGIN
COMMENT
```

```
IF BOTH THE CONTROL AND DATA QUEUES ARE EMPTY, I HAVE NOTHING TO DO, SO
PASSIVATE, ELSE SEE IF THE CONTROL QUEUE HAS ANY PACKETS;
            IF (OUT_MQ.EMPTY AND OUT_CQ.EMPTY) THEN PASSIVATE;
            WHILE (NOT OUT_CQ.EMPTY) DO
                BEGIN
                   PKT :- OUT_CQ.REMOVEQ;
COMMENT
FIND THE PACKET'S DESTINATION AND FILE APPROPRIATELY;
                   IF LOWER(MY,PKT.DEST) THEN
                   BEGIN
                      NEXT_NODE :- NET.NODE(MY.SUCC(PKT.DEST));

COMMENT
FILE IN THE SUCCESSOR'S INPUT DATA QUEUE AND ACTIVATE IT'S RECEIVE
PROCESS;
                      NET.NODE(MY.SUCC(PKT.DEST)).IN_CQ.ADDQ(PKT);
                      ACTIVATE NET.NODE(MY.SUCC(PKT.DEST)).REC_PROCESS;
                   END
                   ELSE BEGIN
COMMENT
FILE IN THE PREDECESSOR'S INPUT CONTROL QUEUE AND ACTIVATE IT'S RECEIVE
PROCESS;
                      NET.NODE(MY.PRED).IN_CQ.ADDQ(PKT);
                      ACTIVATE NET.NODE(MY.PRED).REC_PROCESS;
                   END
                END;
COMMENT
IF THE DATA QUEUE IS NOT EMPTY DO THE FOLLOWING (SIMILAR EXPLANATION AS
ABOVE);
            IF (NOT OUT_MQ.EMPTY) THEN
                BEGIN
                   PKT :- OUT_MQ.REMOVEQ;
                   IF LOWER(MY,PKT.DEST) THEN
                   BEGIN
                      NEXT_NODE :- NET.NODE(MY.SUCC(PKT.DEST));
                      NET.NODE(MY.SUCC(PKT.DEST)).IN_MQ.ADDQ(PKT);
                      ACTIVATE NET.NODE(MY.SUCC(PKT.DEST)).REC_PROCESS;
                   END
                   ELSE BEGIN
                      NET.NODE(MY.PRED).IN_MQ.ADDQ(PKT);
                      ACTIVATE NET.NODE(MY.PRED).REC_PROCESS;
                   END;
                END;
         END ;
   END;


COMMENT
CLASS DEFINITION FOR ALL TYPES OF NODES (APEX, APEX CANDIDATE, ORDINARY).
THE NODES DIFFER ONLY IN THEIR ATTRIBUTES. THIS COMMON STRUCTURE
SIMPLIFIES QUEUE HANDLING. THE ACTIONS TAKEN CORRESPOND TO THE ATTRIB-
UTES.

THE ATTRIBUTES PASSED ARE DATA STORE AND THE OTHERS INDICATE IF THIS NODE
IS THE STARTING APEX OR AN APEX CANDIDATE. THE GLOBAL ADDRESS OF THE NODE
IS ALSO PASSED;

CLASS NODE_CLASS (DATA_STORE,ISAPEX,ISAPEXCAN,ADDR);
COMMENT
POINTER TO THE DATA STORAGE THAT HOLDS THE INPUT DATA,
FLAG INDICATING IF THIS NODE IS AN APEX OR NOT,
FLAG INDICATING IF THIS NODE IS AN APEX CANDIDATE OR NOT, AND
GLOBAL ADDRESS OF THIS NODE;

   REF (DATA_STORAGE) DATA_STORE;
   BOOLEAN ISAPEX,ISAPEXCAN;
   INTEGER ADDR;
```

```
   BEGIN
COMMENT
EACH NODE HAS THREE SETS OF QUEUES. EACH SET CONSISTS OF 1 CONTROL AND
1 DATA QUEUE.
FIRST SET CORRESPONDS TO THE INPUT QUEUES OF THE NODE (IN_MQ, IN_CQ),
SECOND SET CORRESPONDS TO THE TRANSMIT LOWER QUEUES (OUT_LMQ, OUT_LCQ),
THIRD SET CORRESPONDS TO THE TRANSMIT UPPER QUEUES (OUT_UMQ, OUT_UCQ);

      REF (QUEUE) IN_MQ,IN_CQ,OUT_LMQ,OUT_LCQ,OUT_UMQ,OUT_UCQ;
COMMENT
ADDRESS OF THE PREDECESSOR TO THIS NODE,
MESSAGE INTER-ARRIVAL TIME AT THIS NODE,
MESSAGE SERVICE TIME AT THIS NODE (ACTUALLY LINK TRANSMISSION TIME OF THE
LINK CONNECTED TO THIS NODE);

      INTEGER PRED,NO_PKTS;
      REAL IAT;
      REAL SERVICE_TIME;
COMMENT
ROUTING TABLE FOR THIS NODE. IF AN ENTRY IS NON-ZERO THEN ALL PACKETS FOR
THAT INDEX (I.E. NODE HAVING THIS INDEX AS THE GLOBAL ADDRESS) ARE
ROUTED THROUGH THE NODE WHICH HAS ITS ADDRESS AS THIS NON-ZERO ENTRY. IF
ITS ZERO THE PACKETS ARE ROUTED THROUGH THE PREDECESSOR;
      INTEGER ARRAY SUCC(1:NO_NODES);

COMMENT
THIS IS THE TRAFFIC MATRIX. IT CONTAINS THE NUMBER OF PACKETS WHICH A
NODE SENDS ON THE AVERAGE TO ANOTHER NODE;
      REAL    ARRAY FREQ(1:NO_NODES);

COMMENT POINTER TO RECEIVE PROCESS SERVING THIS NODE;
      REF (R_PROCESS) REC_PROCESS;

COMMENT POINTERS TO THE TRANSMIT PROCESSES WHICH ARE SERVING THIS NODE;
      REF (T_PROCESS) TL_PROCESS,TU_PROCESS;

COMMENT POINTER TO THE MESSAGE GENERATOR;
      REF (MESSAGE_GENERATOR) MESG_GEN;

      INTEGER I;

COMMENT CREATION OF THE 6 QUEUES;
      IN_MQ       :- NEW QUEUE;
      IN_CQ       :- NEW QUEUE;
      OUT_LCQ     :- NEW QUEUE;
      OUT_LMQ     :- NEW QUEUE;
      OUT_UCQ     :- NEW QUEUE;
      OUT_UMQ     :- NEW QUEUE;

COMMENT
INITIALIZATION OF THE MESSAGE INTER-ARRIVAL TIME, MESSAGE SERVICE TIME,
AND SOURCE-DESTINATION DISTRIBUTION FOR THIS NODE;
      IAT := 1/DATA_STORE.IAT_TIME(0,ADDR);

      SERVICE_TIME := DATA_STORE.SER_TIME(0,ADDR);

      FREQ(1) := DATA_STORE.TRAFFIC_MTX(0,ADDR,1);
      FOR I := 2 STEP 1 UNTIL NO_NODES  DO
            FREQ(I) := FREQ(I-1)+DATA_STORE.TRAFFIC_MTX(0,ADDR,I);

COMMENT
CREATE AND ACTIVATE A RECEIVE PROCESS, TWO TRANSMIT PROCESSES, AND A
MESSAGE GENERATOR FOR THIS NODE;
      MESG_GEN :- NEW MESSAGE_GENERATOR(THIS NODE_CLASS,ADDR);
      ACTIVATE MESG_GEN;
      REC_PROCESS :- NEW R_PROCESS(THIS NODE_CLASS);
      ACTIVATE REC_PROCESS;
```

```
      TL_PROCESS :- NEW T_PROCESS(THIS NODE_CLASS,OUT_LCQ,OUT_LMQ);
      ACTIVATE TL_PROCESS;
      TU_PROCESS :- NEW T_PROCESS(THIS NODE_CLASS,OUT_UCQ,OUT_UMQ);
      ACTIVATE TU_PROCESS;
END;


COMMENT
THIS IS A CLASS DEFINITION FOR THE MESSAGE GENERATOR. THERE IS ONE
MESSAGE GENERATOR PER NODE. ITS PURPOSE IS TO CREATE DATA MESSAGES
ACCORDING TO THE INTER-ARRIVAL TIME OF THE NODE IT IS SERVING.

IT ALSO PICKS THE DESTINATION NODE ACCORDING TO THE TRAFFIC MATRIX OF THE
NODE IT IS SERVING;

PROCESS CLASS MESSAGE_GENERATOR (MY,NODE_INDEX);

COMMENT THE PARAMETER SPECIFY THE NODE WHICH THIS GENERATOR SERVES;
   INTEGER NODE_INDEX;
   REF (NODE_CLASS) MY;
      BEGIN
         INTEGER I,NEXT;
         REAL RNEXT;
         BOOLEAN NOTFOUND;
         REF (PACKET) PKT;

            WHILE (TRUE) DO
               BEGIN
COMMENT
GENERATE A RANDOM NUMBER BETWEEN 1 AND 1000, SCALE IT TO 100. THIS IS
REQUIRED BECAUSE FREQUENCY CAN HAVE DECIMAL POINTS (SAY 32.6). PICK A
DESTINATION;
                  NEXT := RANDINT(1,1000,U2);
                  RNEXT := NEXT/10.0;
                  I :=0;
                  NOTFOUND := TRUE;
                  WHILE (I<NO_NODES AND NOTFOUND) DO
                     BEGIN
                        I := I+1;
                        IF  RNEXT <=MY.FREQ(I)  THEN NOTFOUND := FALSE;

COMMENT
DO NOT GENERATE IF ADDRESS IS MINE. A NODE DOES NOT SEND ANY MESSAGES
TO ITSELF;
                        IF I=MY.ADDR THEN NOTFOUND := TRUE;
                     END;

COMMENT
WAIT FOR MESSAGE IAT AND SEND A DATA PACKET. ALL PACKETS GENERATED AT A
NODE ARE OF DATA PACKETS;
                  HOLD (NEGEXP(MY.IAT,U1));
                  PKT :- NEW PACKET(NODE_INDEX,I,DATA);
                  PKT.START_TIME:= TIME; COMMENT OUTTEXT("  STR ")
                  OUTFIX(PKT.START_TIME,4,8) OUTIMAGE;
                  MY.IN_MQ.ADDQ(PKT);
                  ACTIVATE MY.REC_PROCESS;
               END;
      END;


COMMENT
THIS PROCESS IS USED FOR TRIGGERING APEX TRANSITIONS. IT INIATIATES A
RECONFIGURATION AT THE SPECIFIED TIME AT WHICH THE TRAFFIC CONDITIONS
CHANGE BY FILING A 'TR' (TRANSITION REQUEST) IN THE INPUT
CONTROL QUEUE OF THE INITIATING NODE OR SOURCE NODE. THIS ACTION IS
REPEATED THE NUMBER OF RECONFIGURATIONS SPECIFIED AT THE START;
```

```
PROCESS CLASS CONFIG_CHANGER(DATA_STORE);

COMMENT POINTER TO THE DATA STORAGE;
        REF (DATA_STORAGE) DATA_STORE;
   BEGIN
      INTEGER SOURCE,NEWAPEX,I,J,K;
      REF (PACKET) PKT;
      FOR K := 1 STEP 1 UNTIL NO_CONFIG DO
        BEGIN

COMMENT
WAIT FOR NEXT TRANSITION TIME (NEXT TIME THE TRAFFIC CONDITIONS CHANGE).
INCREMENT THE NUMBER OF TRANSITIONS (SINCE A TRANSITION IS EFFECTED
LATER ON;
          HOLD(DATA_STORE.CONFIG_TIMES(NUM_TRANSITIONS+1));

          COMMENT OUTINT(NUM_TRANSITIONS,2);
          OUTIMAGE; OUTTEXT("TRAFFIC MTX CHANGES NOW AT TIME");
          OUTFIX(TIME-START_TIME,3,10);
          OUTTEXT(" MILLISECS FROM START TIME");
          OUTIMAGE;

          NUM_TRANSITIONS := NUM_TRANSITIONS + 1;
          COMMENT OUTINT(NUM_TRANSITIONS,2);

COMMENT
READ THE NEW TRAFFIC CONDITIONS (IAT AND THE SOURCE-DESTINATION DISTRI-
BUTION;
          FOR J:= 1 STEP 1 UNTIL NO_NODES DO
            BEGIN
              NET.NODE(J).IAT :=
                              1/DATA_STORE.IAT_TIME(NUM_TRANSITIONS,J);

              NET.NODE(J).FREQ(1) :=
                           DATA_STORE.TRAFFIC_MTX(NUM_TRANSITIONS,J,1);
              COMMENT
                 OUTINT(DATA_STORE.TRAFFIC_MTX(NUM_TRANSITIONS,J,1),4);

              FOR I := 2 STEP 1 UNTIL NO_NODES  DO
                BEGIN
                  NET.NODE(J).FREQ(I) := NET.NODE(J).FREQ(I-1) +
                           DATA_STORE.TRAFFIC_MTX(NUM_TRANSITIONS,J,I);
                  COMMENT
                  OUTINT(DATA_STORE.TRAFFIC_MTX(NUM_TRANSITIONS,J,I),4);
                END;
                COMMENT OUTIMAGE;
            END;

COMMENT
GENERATE A CONTROL PACKET AT THE INITIATING NODE OF TYPE 'TR' AND DATA
FIELD CONTAINING THE NEW APEX TO BE. FILE THIS PACKET IN THE INPUT
CONTROL QUEUE OF THE SOURCE NODE. SEND THIS PACKET TO THE CURRENT APEX
REQUESTING A TRANSITION;

        PKT :- NEW
        PACKET(DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,1),APEX,REQUEST);
        COMMENT
              OUTINT(DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,1),4);

              PKT.START_TIME := TIME;
              PKT.DATA := DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,2);
        COMMENT
              OUTINT(DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,2),4)
              OUTIMAGE;

NET.NODE(DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,1)).IN_CQ.ADDQ(PKT);
```

```
ACTIVATE
NET.NODE(DATA_STORE.INIT_NEWAPEX(NUM_TRANSITIONS,1)).REC_PROCESS;
      END;
   END;



COMMENT
THIS PROCESS IS USED FOR COLLECTING GLOBAL STATISTICS AT SPECIFIC
INTERVALS OF TIME. PRESENTLY IT COLLECTS THE NUMBER OF PACKETS IN DATA
QUEUES OF ACS NODES. THIS SHOULD INCREASE DURING APEX TRANSITION
BECAUSE OF BUFFERING MESSAGES;

PROCESS CLASS DAEMON;
   BEGIN
      INTEGER MQLENGTH,I,J;

      WHILE (TRUE) DO
      BEGIN

COMMENT
SLEEP FOR SOME TIME, COLLECT INFORMATION, PRINT THE INFORMATION AND
SLEEP;
         HOLD(SLEEP_TIME);

         MQLENGTH:=0;

COMMENT THIS IS USED TO PRODUCE OUTPUT IN TWO COLUMNS;
         J := J+1;

         FOR I:=1 STEP 1 UNTIL NO_APEXES DO
             BEGIN
                 MQLENGTH := MQLENGTH + NET.NODE(I).IN_MQ.CARDINAL;
             END;

         OUTTEXT("   ");
         OUTFIX(TIME-START_TIME,3,10);
         OUTINT(MQLENGTH,5);

COMMENT LINE FEED ON EVEN NUMBERED J'S;
         IF((J-2*(J//2))=0) THEN OUTIMAGE;
      END;
   END;


COMMENT
THIS PROCESS IS RESPONSIBLE FOR EXECUTING THE SPECIFIED NUMBER OF BATCHES
FOR THE ENTIRE SIMULATION RUN. IT ENSURES THAT A NEW BATCH BEGINS ONLY
WHEN THE REQUIRED NUMBER OF PACKETS RECEIVED AT THEIR RESPECTIVE
DESTINATION EQUAL THE BATCH SIZE. ALL REQUIRED DATA HAS BEEN COLLECTED
FOR THIS BATCH. AT THE START OF THE NEW BATCH ALL INITIALIZATIONS HAVE
BEEN MADE AND THE VARIOUS PROCESSES ARE ACTIVATED.

PROCESS CLASS BATCH_RUN(UPDATE);

COMMENT POINTER TO THE ACCOUNTING CLASS;
REF (ACCOUNTING) UPDATE;
 BEGIN
      REF (CONFIG_CHANGER) CHANGER;
      REF (DATA_ANALYSIS) STATISTICS;
      INTEGER L,I;

COMMENT
THE BATCHES ARE EXECUTED USING A FOR LOOP. THE FIRST BATCH IS ACTIVATED
WHEN THE NETWORK ACHIEVES STEADY-STATE. AFTER THE SPECIFIED NUMBER OF
OBSERVATIONS HAVE BEEN TAKEN, THE PROCESS IS ACTIVATED;
```

```
      FOR BATCH_COUNT:=1 STEP 1 UNTIL NUM_BATCHES DO BEGIN
      PASSIVATE;

COMMENT
THE MESSAGE GENERATOR, RECEIVE PROCESSES, TRANSMIT LOWER AND UPPER
PROCESSES FOR EACH NODE ARE CANCELLED (TEMPORARILY). THIS IS DONE TO
PROCESS THE ACCUMULATED DATA AFTER A BATCH AND INITIALIZE THE VARIABLES
APPROPRIATELY FOR THE NEXT BATCH. THE CONFIGURATION CHANGER AND THE
DAEMON STATISTICS COLLECTOR ARE ALSO CANCELLED UNTIL THE START OF THE
NEXT BATCH. BEFORE THE START OF A NEW BATCH THE NETWORK IS RECONFIGURED
TO THE FIRST NETWORK TOPOLOGY TTHE TOPOLOGY PRIOR TO ANY
RECONFIGURATIONS);

       FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
           CANCEL (NET.NODE(I).MESG_GEN);
           CANCEL (NET.NODE(I).REC_PROCESS);
           CANCEL (NET.NODE(I).TL_PROCESS);
           CANCEL (NET.NODE(I).TU_PROCESS);
       END;
       CANCEL (NET.COLLECTOR);
       CANCEL (NET.CHANGER); OUTIMAGE;

COMMENT PRINT THE QUEUE LENGTHS OF EACH NODE;
       OUTIMAGE; OUTTEXT("QUEUED PKTS AT EACH NODE :");
       FOR I:=1 STEP 1 UNTIL NO_NODES DO
       OUTINT(NET.NODE(I).IN_MQ.CARDINAL +
              NET.NODE(I).IN_CQ.CARDINAL +
              NET.NODE(I).OUT_LMQ.CARDINAL +
              NET.NODE(I).OUT_LCQ.CARDINAL +
              NET.NODE(I).OUT_UMQ.CARDINAL +
              NET.NODE(I).OUT_UCQ.CARDINAL,3);
      OUTIMAGE; OUTIMAGE;

COMMENT
COMPUTE THE NETWORK THROUGHPUT (THRU_PUT1), NETWORK DELIVERY RATIO
(THRU_PUT2) AND THE AVERAGE DELAY (AVG_DELAY), AND STORE THEM FOR LATER
ANALYSIS;
      THRU_PUT1 := THRU_PUT2 := AVG_DELAY := 0.0;
      THRU_PUT1 := SEQNR/(TIME-START_TIME);
      THRU_PUT2 := RECEIVED/SEQNR;
      AVG_DELAY := TOTAL_DELAY/RECEIVED;
      UPDATE.MEAN_DELAY(BATCH_COUNT):=AVG_DELAY;
      UPDATE.THROUGHPUT(BATCH_COUNT):=THRU_PUT1;
      UPDATE.NET_RATIO(BATCH_COUNT):=THRU_PUT2;
      OUTIMAGE; OUTIMAGE;

COMMENT
PRINT SOME USEFUL INFORMATION FOR THIS BATCH;
      OUTI("TRANSITIONS OCCURED  : ",NUM_TRANSITIONS);
      OUTI("PKTS GENERATED       : ",SEQNR);
      OUTI("RECEIVED             : ",RECEIVED);
      OUTI("PACKETS IN SYSTEM    : ",SEQNR-RECEIVED);
      OUTTEXT("SIMULATION TIME      : ");
      OUTFIX(TIME-START_TIME,6,18); OUTIMAGE;
      OUTTEXT("TOTAL_DELAY          : ");OUTFIX(TOTAL_DELAY,6,18);
         OUTTEXT(" MILLISECONDS"); OUTIMAGE;
      OUTTEXT("AVERAGE DELAY        : ");
                OUTFIX(UPDATE.MEAN_DELAY(BATCH_COUNT),6,18);
         OUTTEXT(" MILLISECONDS"); OUTIMAGE;
      OUTTEXT("THROUGHPUT           : ");OUTFIX(THRU_PUT1,6,18);
         OUTTEXT(" PKTS/MILLISECONDS"); OUTIMAGE;
      OUTTEXT("NET DELIVERY RATIO   : ");OUTFIX(THRU_PUT2,6,18);
         OUTIMAGE;
      OUTIMAGE;
      OUTTEXT("LAST  U1 :"); OUTINT(U1,20); OUTIMAGE;
      OUTTEXT("LAST  U2 :"); OUTINT(U2,20); OUTIMAGE;
```

```
COMMENT ANALYSE THE SIMULATION RESULTS AFTER 15 BATCHES;
      IF (BATCH_COUNT=15) THEN BEGIN
         STATISTICS :- NEW DATA_ANALYSIS(UPDATE);
         ACTIVATE STATISTICS;
         END;

COMMENT
ACTIVATE DATA ANALYSIS OF THE SIMULATION RESULTS AFTER 5, 10, 20,
40 BATCHES;
      IF (BATCH_COUNT-5*INT=0) THEN BEGIN
         INT:=INT*2;
         STATISTICS :- NEW DATA_ANALYSIS(UPDATE);
         ACTIVATE STATISTICS;
         END;
      OUTIMAGE;  OUTIMAGE;

COMMENT
AFTER THE ACCUMULATED DATA HAS BEEN PROCESSED, THE NEW BATCH IS STARTED
IF THE BATCH COUNTER HAS NOT EXCEEDED THE SPECIFIED NUMBER. INITIALIZE
THE VARIABLES BEFORE THE NEW BATCH STARTS. SEQNR REPRESENTS THE NUMBER OF
PACKETS IN THE SYSTEM. THIS IS UPDATED TO THE NUMBER GENERATED IN THE
SYSTEM IN THE PREVIOUS BATCH MINUS THE NUMBER OF PACKETS RECEIVED IN THAT
BATCH;


      IF BATCH_COUNT <> NUM_BATCHES THEN BEGIN
      OUTTEXT("BATCH : "); OUTINT(BATCH_COUNT+1,3); OUTIMAGE;


      NUM_TRANSITIONS  := 0;
      TRANSITION       := FALSE;
      SEQNR            := SEQNR - RECEIVED;
      RECEIVED         := 0;
      TOTAL_DELAY      := 0.0;
      START_TIME       := TIME;
      OUTIMAGE;
      OUTTEXT("FIRST U1 :"); OUTINT(U1,20); OUTIMAGE;
      OUTTEXT("FIRST U2 :"); OUTINT(U2,20); OUTIMAGE;
      OUTTEXT("START TIME :");  OUTFIX(START_TIME,6,18); OUTIMAGE;

COMMENT
RECONSTRUCT THE ORIGINAL NETWORK CONFIGURATION AT START OF THIS BATCH SO
THAT THIS BATCH UNDERGORES THE SAME RECONFIGURATIONS AS THE EARLIER ONE.
1. MAKE THE STARTING APEX (NODE 1) AS THE NETWORK APEX.
2. INITIALIZE THE CONNECTION MATRIX.
3. INITIALIZE THE TOPOLOGY INFORMATION.
4. INITIALIZE THE NUMBER OF PACKETS AT EACH NODE, MESSAGE INTERARRIVAL
   TIME, SERVICE TIME, SOURCE DESTINATION DISTRIBUTION AT EACH NODE;
5. CREATE THE INITIAL ROUTING TABLE WITH RESPECT TO THE STARTING APEX
   (NODE 1);

      NET.E; NET.A; NET.B; NET.D; NET.ZEROING; NET.BUILD(APEX,0);
      OUTIMAGE;  OUTIMAGE;

COMMENT
ACTIVATE EACH NODE'S MESSAGE GENERATOR, RECEIVE PROCESS, TRANSMIT LOWER
AND UPPER PROCESS NOW;
      FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
          ACTIVATE (NET.NODE(I).MESG_GEN);
          ACTIVATE (NET.NODE(I).REC_PROCESS);
          ACTIVATE (NET.NODE(I).TL_PROCESS);
          ACTIVATE (NET.NODE(I).TU_PROCESS);
      END;

COMMENT CREATE A CONFIGURATION CHANGER AND ACTIVATE IT;
      CHANGER:- NEW CONFIG_CHANGER(DATA_STORE);
      ACTIVATE CHANGER;
```

```
COMMENT ACTIVATE THE DAEMON STATISTICA COLLECTOR;
      ACTIVATE (NET.COLLECTOR);
     END;
  END;

COMMENT ACTIVATE THE MAIN PROGRAM AFTER ALL BATCHES HAVE RUN;
 ACTIVATE MAIN;
 END;


COMMENT
THIS IS THE CLASS DEFINITION OF THE WHOLE NETWORK. IT INTEGERATES ALL THE
SUB COMPONENTS OF THE NETWORK. ONLY ONR INSTANCE OF THIS OBJECT EXISTS
AT A TIME. IT PUTS ALL THE NODES INTO A SINGLE NETWORK, BY THE TOPOLOGY
SPECIFIED (CONNECTION MATRIX). IT CREATES SYSTEM WIDE PROCESSES LIKE
NEW_BATCH, CONFIGURATION_CHANGER AND DAEMON;

CLASS NETWORK(DATA_STORE);
COMMENT POINTER TO THE DATA STORAGE THAT HOLDS ALL THE INPUT DATA;
      REF (DATA_STORAGE) DATA_STORE;

  BEGIN
COMMENT
ARRAY OF NODES COMPRISING THIS NETWORK. THERE ARE 'NO_NODES' NODES AND
'NO_APEXES' APEX CANDIDATES;
      REF (NODE_CLASS) ARRAY NODE(1:NO_NODES);

COMMENT
THIS MATRIX SPECIFIES THE NETWORK CONNECTIVITY. IF AN ENTRY IS TRUE THEN
THE FIRST INDEX (NODE) IS CONNECTED BY A LINK TO THE SECOND INDEX (NODE);
      BOOLEAN ARRAY CONNECTED(1:NO_NODES,1:NO_NODES);
COMMENT
POINTERS TO THE NEW_BATCH, CONFIGURATION CHANGER AND DAEMON STATISTICS
COLLECTOR;
      REF (BATCH_RUN) NEW_BATCH;
      REF (CONFIG_CHANGER) CHANGER;
      REF (DAEMON) COLLECTOR;

      INTEGER I,J,K;
COMMENT PREDECESSOR NODE AND SUCCESSOR NODE;
      INTEGER PREDNODE,SUCCNODE;

COMMENT
THIS IS A RECURSIVE PROCEDURE AND IS USED TO CREATE THE ROUTING TABLE
AT THE START OF THE SIMULATION RUN AND FOR TABLE REARRANGEMENT DURING
TRANSITION;
  PROCEDURE BUILD(NODE_INDEX,PREV);
COMMENT NODE_INDEX IS THE ROOT OF THE SUB-TREE;
      INTEGER NODE_INDEX,PREV;
      BEGIN
         INTEGER I,J,SUCCNODE;

         FOR I:= 1 STEP 1 UNTIL NO_NODES DO
         BEGIN
COMMENT
IF ROOT AND 'I' ARE CONNECTED AND IS NOT PREVIOUS ROOT;
            IF CONNECTED(NODE_INDEX,I) AND (NOT I=PREV) THEN
               BEGIN
COMMENT
MAKE NODE 'I' SUCCESSOR OF ROOT OR 'NODE_INDEX', PIVOT AROUND THE
SUB-TREE WITH ROOT 'I', MAKE THE ROOT (NODE_INDEX) PREDECESSOR OF 'I',
MAKE ALL SUCESSORS OF 'I',SUCESSORS OF ROOT OR 'NODE_INDEX';
                  NODE(NODE_INDEX).SUCC(I) := I;
                  BUILD(I,NODE_INDEX);
                  NODE(I).PRED := NODE_INDEX;
                  FOR J:= 1 STEP 1 UNTIL NO_NODES DO
```

```
                    IF NODE(I).SUCC(J)>0  THEN
                       NODE(NODE_INDEX).SUCC(J):=I;
                END;
          END;
 END ****BUILD****;

COMMENT
FOR DEBUGGING PURPOSES. USED TO PRINT THE SUCESSOR MATRIX WHEN TRANSI-
TION TAKES PLACE;
 PROCEDURE PRINTROUTINE;
 BEGIN
          INTEGER I,J;
          FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
              FOR J:=1 STEP 1 UNTIL NO_NODES DO
              OUTINT(NODE(I).SUCC(J),2);
              OUTIMAGE;
              END;
 END ****PRINTROUTINE***;

COMMENT
ZEROES OUT THE PRESENT SUCESSOR MATRIX. THIS IS DONE BEFORE CREATING
THE NEW ROUTING TABLE DURING TRANSITION;
 PROCEDURE ZEROING;
 BEGIN
          INTEGER I,J;
           FOR I:=1 STEP 1 UNTIL NO_NODES DO
               FOR J:=1 STEP 1 UNTIL NO_NODES DO
               NODE(I).SUCC(J):=0;
 END ****ZEROING***;

COMMENT
FOR DEBUGGING PURPOSES. FOR EACH NODE PRINT '1' IF NODE IS AN APEX AND
ALSO PRINT '1' IF NODE IS AN APEX CANDIDATE SET NODE;
 PROCEDURE CHECK;
 BEGIN
          INTEGER I;
          FOR I:= 1 STEP 1 UNTIL NO_NODES DO BEGIN
              IF NODE(I).ISAPEX THEN
                  OUTTEXT("1") ELSE OUTTEXT("0");
              IF NODE(I).ISAPEXCAN THEN
                  OUTTEXT("1") ELSE OUTTEXT("0");
              OUTIMAGE;
              END;
 END ***CHECK***;


COMMENT
DURING TRANSITION READ THE NEW SERVICE TIMES FOR ALL NODES.
NUM_TRANSITION INDICATES THE SERVICE TIMES CORRESPONDING TO THIS TRAFFIC
CONDITION;

 PROCEDURE READ_SER_PRNT;
 BEGIN
              INTEGER I;
              OUTIMAGE;
              FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
                  NODE(I).SERVICE_TIME:=
                  DATA_STORE.SER_TIME(NUM_TRANSITIONS,I);
                  COMMENT OUTFIX(NODE(I).SERVICE_TIME,4,7);
              END;
              COMMENT  OUTIMAGE;
     END  ****SERPRINT*** ;


COMMENT
THIS PROCEDURE IS CALLED AT THE TIME STEADY STATE CONDITIONS ARE OBTAINED
```

```
IT REINITIALIZES THE QUEUE VARIABLES TO ZERO AND START TIME TO THE
CURRENT TIME;
 PROCEDURE REINITIALIZE_DELETE_STATISTICS;
 BEGIN
      INTEGER I,J,TEMP;

      OUTIMAGE;
      OUTTEXT("START OF STEADY STATE SIMULATION AFTER "); OUTIMAGE;
      OUTINT(RECEIVED,5); OUTTEXT(" PKTS DELIVERED IN TIME");
      OUTFIX(TIME-START_TIME,3,10); OUTIMAGE; OUTIMAGE;

COMMENT PRINT THE TOTAL QUEUE LENGTHS OF EACH NODE;
      OUTTEXT("QUEUE CONTENTS FOR ALL QUEUES FOR NODES 1 TO ");
      OUTINT(NO_NODES,2); OUTIMAGE;

          FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
              TEMP := 0;
              TEMP := TEMP + NODE(I).IN_MQ.CARDINAL +
                             NODE(I).OUT_UMQ.CARDINAL +
                             NODE(I).OUT_LMQ.CARDINAL;
              OUTINT(TEMP,5);

COMMENT INITIALIZE THE QUEUE VARIABLES TO ZERO;
              NODE(I).IN_MQ.MAXLEN:=0;   NODE(I).IN_MQ.TIME_X_N:=0.0;
              NODE(I).IN_CQ.MAXLEN:=0;   NODE(I).IN_CQ.TIME_X_N:=0.0;
              NODE(I).OUT_UMQ.MAXLEN:=0; NODE(I).OUT_UMQ.TIME_X_N:=0.0;
              NODE(I).OUT_LMQ.MAXLEN:=0; NODE(I).OUT_LMQ.TIME_X_N:=0.0;
              NODE(I).OUT_UCQ.MAXLEN:=0; NODE(I).OUT_UCQ.TIME_X_N:=0.0;
              NODE(I).OUT_LCQ.MAXLEN:=0; NODE(I).OUT_LCQ.TIME_X_N:=0.0;

              END;

          OUTIMAGE;
 COMMENT  OUTTEXT("TIME :") OUTFIX(TIME,3,10) OUTTEXT("  REC :")
          OUTINT(RECEIVED,6);
 COMMENT  OUTTEXT("    SEQ :") OUTINT(SEQNR,5)
          OUTTEXT("    REC :") OUTINT(RECEIVED,5);

COMMENT INITIALIZE THE VARIOUS SYSTEM VARIABLES;
          SEQNR        :=SEQNR-RECEIVED;
          RECEIVED     :=0;
          TOTAL_DELAY  :=0.0;
          START_TIME   :=TIME;
          STEADY_STATE :=TRUE;  COMMENT MAKE STEADY STATE TRUE;

          COMMENT  OUTIMAGE
          OUTTEXT(" TOT_DEL :") OUTFIX(TOTAL_DELAY,3,10)
          OUTTEXT(" START   :") OUTFIX(START_TIME,3,10)  OUTIMAGE;

COMMENT START THE FIRST BATCH;
          OUTTEXT("BATCH :  1"); OUTIMAGE;
          OUTTEXT("FIRST U1 :"); OUTINT(U1,20); OUTIMAGE;
          OUTTEXT("FIRST U2 :"); OUTINT(U2,20); OUTIMAGE;

COMMENT CREATE A NEW BATCH AND ACTIVATE IT;
          NEW_BATCH :- NEW BATCH_RUN(UPDATE);
          ACTIVATE NEW_BATCH;

COMMENT
CREATE A NEW CONFIGURATION CHANGER AND DAEMON STATISTICS COLLECTOR (AT
THE START OF DATA COLLECTION) AND ACTIVATE THEM;
          CHANGER :- NEW CONFIG_CHANGER(DATA_STORE) ;
          ACTIVATE CHANGER;

          COLLECTOR :- NEW DAEMON;
          ACTIVATE COLLECTOR;
 END *****REINIT_DEL_STAT***;
```

```
COMMENT INITIALIZE THE CONNECTION MATRIX;
PROCEDURE A;
BEGIN
        INTEGER I,J;
        FOR I:=1 STEP 1 UNTIL NO_NODES DO
            FOR J:=1 STEP 1 UNTIL NO_NODES DO
                CONNECTED(I,J) := FALSE;
END;


COMMENT
THIS PROCEDURE CREATES THE CONNECTION MATRIX. THE FIRST NODE IN THE
NETWORK CONNECTIVITY INFORMATION IS ASSUMED TO BE THE APEX;

PROCEDURE B;
BEGIN
     INTEGER K;
     APEX := DATA_STORE.NET_LINKS(1);
     PREDNODE := APEX;
     K:=2;
     WHILE (NOT PREDNODE = -1) DO
        BEGIN
           SUCCNODE := DATA_STORE.NET_LINKS(K);
           CONNECTED(PREDNODE,SUCCNODE) := TRUE;
           CONNECTED(SUCCNODE,PREDNODE) := TRUE;
           PREDNODE := DATA_STORE.NET_LINKS(K+1);
           K:=K+2;
        END;
END;


COMMENT
THIS PROCEDURE CREATES AN APEX, APEX CANDIDTES AND ORDINARY NODES;
PROCEDURE C;
BEGIN
     NODE(APEX) :- NEW NODE_CLASS(DATA_STORE,TRUE,TRUE,APEX);

     FOR I:=1 STEP 1 UNTIL NO_APEXES DO
     IF NOT I=APEX THEN NODE(I) :-
                NEW NODE_CLASS(DATA_STORE,FALSE,TRUE,I);

     FOR I := NO_APEXES+1 STEP 1 UNTIL NO_NODES DO
           NODE(I) :- NEW NODE_CLASS(DATA_STORE,FALSE,FALSE,I);
END;


COMMENT
THIS PROCEDURE IS USED AT THE START OF A NEW BATCH TO CREATE THE INITIAL
TRAFFIC SCENARIO (SOURCE DESTINATION DISTRIBUTION) AND ASSIGN THE CORRES-
PONDING MESSAGE INTER-ARRIVAL TIMES AND SERVICE TIMES TO ALL THE NODES.

PROCEDURE D;
BEGIN
   INTEGER I,J,TEMP;

   FOR I:= 1 STEP 1 UNTIL NO_NODES DO BEGIN
       TEMP:=0;
       TEMP:=   NODE(I).IN_MQ.CARDINAL +
                    NODE(I).IN_CQ.CARDINAL +
                    NODE(I).OUT_UMQ.CARDINAL +
                    NODE(I).OUT_LMQ.CARDINAL +
                    NODE(I).OUT_UCQ.CARDINAL +
                    NODE(I).OUT_LCQ.CARDINAL;
COMMENT
AS A ROUGH MEASURE THE NUMBER OF PACKETS QUEUED AT A NODE ARE ASSIGNED
```

```
 TO THE NUMBER OF PACKETS GENERATED AT THAT NODE AT THE START OF A NEW
 BATCH. IT SHOULD BE NOTED THAT THIS INFORMATION IS NOT BEING USED IN
 ANY DATA ANALYSIS;
         NODE(I).NO_PKTS := TEMP;
         COMMENT OUTINT(TEMP,5) ;
         COMMENT OUTIMAGE;

         NODE(I).IAT := 1/DATA_STORE.IAT_TIME(0,I);
         NODE(I).SERVICE_TIME := DATA_STORE.SER_TIME(0,I);
         NODE(I).FREQ(1) := DATA_STORE.TRAFFIC_MTX(0,I,1);
         FOR J:= 2 STEP 1 UNTIL NO_NODES DO
             NODE(I).FREQ(J) := NODE(I).FREQ(J-1) +
                             DATA_STORE.TRAFFIC_MTX(0,I,J);
         END;
    END;


COMMENT
THIS PROCEDURE IS USED AT THE START OF A NEW BATCH. THE EXISTING APEX
AT THE END OF A BATCH IS MADE FALSE AND THE APEX AT THE START OF THE RUN
(FIRST NODE IN THE NETWORK CONNECTIVITY INFORMATION) IS MADE THE CURRENT
APEX.
 PROCEDURE E;
 BEGIN
 NODE(APEX).ISAPEX := FALSE;
 APEX := DATA_STORE.NET_LINKS(1);
 NODE(APEX).ISAPEX := TRUE;
 END;


COMMENT INITIALIZE THE CONNECTION MATRIX;
      A;
COMMENT READ THE CONNECTION INFORMATION;
      B;
COMMENT CREATE AN APEX, APEX CANDIDATES AND ORDINARY NODES;
      C;
COMMENT INITIALIZE THE SUCCESSOR MATRIX;
      ZEROING;
COMMENT
CREATE THE INITIAL ROUTING TABLES, PIVOTING IT AROUND THE STARTING APEX
(FIRST NODE IN THE NETWORK CONNECTION INFORMATION);
      BUILD(APEX,0);
      COMMENT PRINTROUTINE;
      COMMENT CHECK;
      COMMENT OUTIMAGE;
   END;




COMMENT
THIS PROCEDURE IS USED FOR DEBUGGING PURPOSES AND ALSO TRACING THE EVENTS
OCCURRING DURING THE SIMULATION RUN. GIVEN A QUEUE IT WILL SPEW OUT ALL
THE INFORMATION AND THE PACKETS IN THE QUEUE;

PROCEDURE PRINTQ (Q);
COMMENT THE QUEUE TO BE PRINTED OUT;
   REF (QUEUE) Q;
   BEGIN
      REF (PACKET) N;
COMMENT LOOK AT THE FIRST PACKET IN THE QUEUE;
      N :- Q.FIRST;
COMMENT PRINT OUT ALL QUEUE STATISTICS;
      OUTTEXT("QUEUE : "); OUTINT(Q.ADDR,2); OUTIMAGE;
      OUTR("QUEUE INFO AT   ",TIME);
      OUTR("   LASTTIME     ",Q.LAST_TIME);
```

```
      OUTR("   TIME_X_N      ",Q.TIME_X_N);
      OUTI("   MAXLEN        ",Q.MAXLEN);
      OUTI("   TOTAL         ",Q.TOTAL);

COMMENT
FIND OUT THE AVERAGE TIME SPENT BY A PACKET IN THE QUEUE AND PRINT IT;
      IF TIME > 0.0 THEN
         OUTR("   TIME_AVG_LEN ",Q.TIME_X_N/TIME);
      OUTI("   NO OF PACKETS",Q.CARDINAL);
      WHILE ( N =/= NONE ) DO
         BEGIN
COMMENT SPEW OUT ALL THE INFORMATION FOR ALL THE PACKETS IN THIS QUEUE;

      OUTTEXT(" SRC "); OUTINT(N.SRC,4); OUTTEXT(" DEST ");
      OUTINT(N.DEST,4); OUTTEXT(" TYPE"); OUTINT(N.TYPE,4);
      COMMENT OUTTEXT(" DATA") OUTINT(N.DATA,4);
      OUTTEXT("START "); OUTFIX(N.START_TIME,3,10);
            OUTIMAGE;
            N :- N.NEXT;
         END;
      OUTIMAGE;
   END;


COMMENT
THIS CLASS STORES ALL THE INPUT DATA REQUIRED FOR A BATCH RUN. THIS WAS
DONE TO AVOID REWRITING THE INPUT DATA AS MANY TIMES AS THE NUMBER OF
BATCHES SPECIFIED IN THE SIMULATION RUN;

CLASS DATA_STORAGE;
BEGIN
    INTEGER I,J,K;

COMMENT
NUM_ENTRIES SPECIFIES THE NUMBER OF ENTRIES IN THE INPUT DATA REGARDING
THE NETWORK CONNECTIVITY INFORMATION. THIS ALSO INCLUDES THE END OF
CONNECTION DATA SET MARKER -1. ARRAY NET_LINKS HOLDS THE POINT-TO-POINT
CONNECTION BETWEEN THE VARIOUS NODES;
    INTEGER ARRAY NET_LINKS(1:NUM_ENTRIES);

COMMENT
THIS HOLDS INFORMATION REGARDING THE INITIATING NODE AND THE NEWAPEX
FOR EACH RECONFIGURATION. NO_CONFIG DENOTES THE NUMBER OF RECONFIGURATION
A NETWORK UNDERGOES IN EACH BATCH;
    INTEGER ARRAY INIT_NEWAPEX(1:NO_CONFIG,1:2);

COMMENT THIS HOLDS THE TIME LEFT FOR NEXT TRANSITION IN ORDER;
    REAL ARRAY CONFIG_TIMES(1:NO_CONFIG);

COMMENT
SER_TIME HOLDS THE SERVICE TIMES FOR ALL NODES UNDER THE DIFFERENT
TRAFFIC CONDITIONS. IAT_TIME HOLDS THE MESSAGE INTER-ARRIVAL TIME
FOR ALL NODES UNDER THE DIFFERENT TRAFFIC CONDITIONS. IT SHOULD BE NOTED
THAT NO_CONFIG = NUMBER OF TRAFFIC CONDITIONS - 1, HENCE THE SUBSCRIPT 0.
SUBSCRIPT 0 DENOTES THE INITIAL NETWORK CONDITIONS AT THE START OF THE
SIMULATION RUN AND EACH BATCH;
    REAL ARRAY SER_TIME(0:NO_CONFIG,1:NO_NODES);
    REAL ARRAY IAT_TIME(0:NO_CONFIG,1:NO_NODES);

COMMENT THIS HOLDS THE TRAFFIC MATRICES UNDER THE DIFFERENT SCENARIOS;
    REAL ARRAY TRAFFIC_MTX(0:NO_CONFIG,1:NO_NODES,1:NO_NODES);

COMMENT READ THE NETWORK CONNECTION INFORMATION;
       FOR I:=1 STEP 1 UNTIL NUM_ENTRIES DO
           NET_LINKS(I) := ININT;
```

```
COMMENT
READ IN THE IATS, SERVICE TIMES AND TRAFFIC MATRIX FOR THE INITIAL
NETWORK SCENARIO. THE NODES ARE INITIALIZED WITH THESE VALUES AT THE
START OF EACH BATCH;
       FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
              IAT_TIME(0,I):= INREAL;
              SER_TIME(0,I):= INREAL;

              COMMENT OUTFIX(IAT_TIME(0,I),4,10);
              COMMENT OUTFIX(SER_TIME(0,I),4,10);

           FOR J:=1 STEP 1 UNTIL NO_NODES DO BEGIN
              TRAFFIC_MTX(0,I,J):=INREAL;
              COMMENT OUTINT(TRAFFIC_MTX(0,I,J),3);
              END; COMMENT OUTIMAGE;
       END;

COMMENT
FOR THE NUMBER OF RECONFIGURATIONS SPECIFIED READ
1. TIME LEFT FOR NEXT TRANSITION (TIME AT WHICH THE NEW TRAFFIC MATRIX
   FOLLOWING THIS INFORMATION IS READ IN).
2. INITIATING NODE AND NEW APEX FOR THIS TRAFFIC CONDITION.
3. IAT, SERVICE TIME AND SOURCE DESTINATION DISTRIBUTION OF THIS NEW
   TRAFFIC MATRIX;

       FOR K:=1 STEP 1 UNTIL NO_CONFIG DO BEGIN
           CONFIG_TIMES(K):= INREAL;
           INIT_NEWAPEX(K,1):= ININT;
           INIT_NEWAPEX(K,2):= ININT;

           COMMENT OUTFIX(CONFIG_TIMES(K),4,10);
           COMMENT OUTINT(INIT_NEWAPEX(K,1),4);
           COMMENT OUTINT(INIT_NEWAPEX(K,2),4)
           OUTIMAGE;

           FOR I:=1 STEP 1 UNTIL NO_NODES DO BEGIN
               IAT_TIME(K,I):= INREAL;
               SER_TIME(K,I):= INREAL;

               COMMENT OUTFIX(IAT_TIME(K,I),4,10);
               COMMENT OUTFIX(SER_TIME(K,I),4,10);

               FOR J:=1 STEP 1 UNTIL NO_NODES DO BEGIN
                   TRAFFIC_MTX(K,I,J):=INREAL;
                   COMMENT OUTINT(TRAFFIC_MTX(K,I,J),3);
                   END;
                   COMMENT OUTIMAGE;
              END;
           END;
END;


COMMENT
THIS CLASS DEFINES THE ARRAYS FOR THE DIFFERENT NETWORK PERFORMANCE
MEASURES. THEY HOLD THE VALUES FOR THE PERFORMANCE MEASURES FOR EACH
BATCH.
1. MEAN DELAY = TOTAL NETWORK DELAY / NUMBER OF PACKETS RECEIVED.
2. THROUGHPUT = NUMBER OF PACKETS RECEIVED / TOTAL BATCH SIMULATION TIME.
3. NET DELIEVERY RATIO = NUMBER OF PACKETS RECEIVED / NUMBER OF PACKETS
                                                              GENERATED;
 CLASS ACCOUNTING;
   BEGIN
       LONG REAL ARRAY MEAN_DELAY(1:NUM_BATCHES);
       LONG REAL ARRAY THROUGHPUT(1:NUM_BATCHES);
       LONG REAL ARRAY  NET_RATIO(1:NUM_BATCHES);
   END;
```

```
COMMENT
THIS PROCESS DEALS WITH THE ANALYSIS OF SIMULATION RESULTS. FOR EACH
NETWORK PARAMETER, IT COMPUTES THE OVERALL MEAN, VARIANCE, AUTOCOVARIANCE
AUTOCORRELATION BETWEEN THE BATCH MEANS AND 95% CONFIDENCE INTERVAL;

PROCESS CLASS DATA_ANALYSIS(UPDATE);
COMMENT
POINTER TO THE CLASS ACCOUNTING WHICH HOLDS THE REQUIRED INFORMATION;
      REF (ACCOUNTING) UPDATE;
BEGIN

COMMENT
THIS PROCEDURE PRINTS THE BATCH MEANS FOR THE THREE PERFORMANCE MEASURES;

PROCEDURE TITLE(CIDELAY,CITHRU,CINET);
COMMENT
THE THREE ARRAYS DEFINED IN CLASS ACCOUNTING ARE PASSED AS PARAMETERS;
LONG REAL ARRAY CIDELAY, CITHRU, CINET;
BEGIN
    INTEGER I;
    EJECT(1); OUTIMAGE; OUTIMAGE; OUTIMAGE;
    OUTTEXT("ANALYSIS OF SIMULATION RESULTS");
    OUTTEXT(" FOR THE METHOD OF BATCH MEANS"); OUTIMAGE;
    OUTTEXT("-----------------------------");
    OUTTEXT("-----------------------------"); OUTIMAGE;
    OUTIMAGE; OUTIMAGE;
    OUTINT(NO_NODES,2);
    OUTTEXT("  NODE DHLAN STUDY"); OUTIMAGE; OUTIMAGE;
    OUTTEXT("NUMBER OF BATCHES : ");
    OUTINT(NO_BATCHES,3); OUTIMAGE; OUTIMAGE;
    OUTTEXT("NO_APEX_CANDIDATES  ");
OUTTEXT("NO_TRANSITIONS  TRANSITION_TIME");
OUTIMAGE;
OUTTEXT("         ");
OUTINT(NO_APEXES,1); OUTTEXT("                        ");
OUTINT(NUM_TRANSITIONS,1); OUTTEXT("          ");
OUTFIX(TRANSIT_TIME,1,5); OUTTEXT(" MILLISECS"); OUTIMAGE; OUTIMAGE;
OUTTEXT("BATCH MEANS FOR THE NETWORK PARAMETERS");  OUTIMAGE;
OUTTEXT("**************************************");  OUTIMAGE;
OUTIMAGE;
OUTTEXT(" BATCH     AVERAGE DELAY");
OUTTEXT("     THROUGHPUT    NET DELIVERY RATIO");
OUTIMAGE;
OUTTEXT("              MILLISECS"); OUTIMAGE;
OUTIMAGE; OUTIMAGE;
FOR I := 1 STEP 1 UNTIL NO_BATCHES DO
BEGIN
    OUTINT(I,4); OUTTEXT("         ");
    OUTFIX(CIDELAY(I),3,8); OUTTEXT("            ");
    OUTFIX(CITHRU(I),3,5); OUTTEXT("              ");
    OUTFIX(CINET(I),5,8);
    OUTIMAGE;
END;
END ***TITLE***;



COMMENT
THIS PROCEDURE COMPUTES THE OVERALL MEAN AND COVARIANCE BETWEEN THE
BATCH MEANS;
PROCEDURE XBAR_COV_STAT(CIDELAY,CITHRU,CINET);
LONG REAL ARRAY CIDELAY, CITHRU, CINET;
BEGIN
```

```
COMMENT
THIS HOLDS THE COVARIANCE MATRIX FOR THE THREE PERFORMANCE MEASURES;
    LONG REAL ARRAY COV(1:3,1:NO_BATCHES,1:NO_BATCHES);

COMMENT
R(I,J) HOLDS THE VALUE OF THE SUM OF [COV(I,K,K+J)] WHERE J VARIES FROM
0 TO NUMBER OF BATCHES - 1 AND K VARIES FROM 1 TO NUMBER OF BATCHES - J.
INDEX I (0, 1, 2) STANDS FOR COVARIANCE MATRICES FOR AVGDELAY, THROUGHPUT
AND NETWORK DELIVERY RATIO;
    LONG REAL ARRAY R(0:2,0:NO_BATCHES-1);

COMMENT THIS HOLDS THE OVERALL MEANS FOR THE THREE PERFORMANCE MEASURES
IN THE ORDER AVG_DELAY, THROUGHPUT AND NETWORK DELIVERY RATIO. THE
SUBSCRIPT 0 AND 1 AT DIFFERENT PLACES IS CONFUSING. SORRY ABOUT THAT;
    LONG REAL ARRAY XBAR(1:3);

COMMENT COMPUTE THE OVERALL MEANS;
    LONG REAL TEMP1, TEMP2, TEMP3;
    INTEGER J,K;
    TEMP1 := TEMP2 := TEMP3 := 0.0;
    FOR J:=1 STEP 1 UNTIL NO_BATCHES DO BEGIN
        TEMP1 := TEMP1 + CIDELAY(J);
        TEMP2 := TEMP2 + CITHRU(J);
        TEMP3 := TEMP3 + CINET(J);
        END; OUTIMAGE;
    XBAR(1) :=TEMP1/NO_BATCHES;
    XBAR(2) :=TEMP2/NO_BATCHES;
    XBAR(3) :=TEMP3/NO_BATCHES;

    COMMENT COVARIANCE BETWEEN THE MEANS;
    FOR J:= 1 STEP 1 UNTIL NO_BATCHES DO BEGIN
    FOR K:= J STEP 1 UNTIL NO_BATCHES DO BEGIN
        COV(1,J,K):=COV(1,K,J):=(CIDELAY(J)-XBAR(1))*
                                (CIDELAY(K)-XBAR(1));
        COV(2,J,K):=COV(2,K,J):=(CITHRU(J)-XBAR(2))*
                                (CITHRU(K)-XBAR(2));
        COV(3,J,K):=COV(3,K,J):=(CINET(J)-XBAR(3))*
                                (CINET(K)-XBAR(3));
        END;
    END;
    FOR J:=0 STEP 1 UNTIL NO_BATCHES-1 DO
    BEGIN
        TEMP1 := TEMP2 := TEMP3 := 0.0;
        FOR K:=1 STEP 1 UNTIL NO_BATCHES-J DO BEGIN
            TEMP1 := TEMP1 + COV(1,K,K+J);
            TEMP2 := TEMP2 + COV(2,K,K+J);
            TEMP3 := TEMP3 + COV(3,K,K+J);
            END;
        R(0,J):= TEMP1;  COMMENT FOR MEAN DELAY;
        R(1,J):= TEMP2;  COMMENT FOR THROUGHPUT;
        R(2,J):= TEMP3;  COMMENT FOR NETWORK DELIVERY RATIO;
    END;

COMMENT PRINT THE STATISTICS COMPUTED ABOVE;
    STAT_PRINT(XBAR,COV,R);
COMMENT
CALL THE CORRELATION PROCEDURE AND THE CONFIDENCE INTERVAL COMPUTATIONS
PROCEDURE;
    CORELATION(CIDELAY,CITHRU,CINET,R,XBAR);
    BATCH_CONFINT(XBAR,R);
END   ***XBARCOVSTAT***;



COMMENT
PRINT STATISTICS FOR THE OVERALL MEAN, COVARIANCE MATRIX AND AUTOCOVARIA-
NCE FUNCTION;
```

```
PROCEDURE STAT_PRINT(XBAR,COV,R);
LONG REAL ARRAY XBAR; LONG REAL ARRAY COV; LONG REAL ARRAY R;
BEGIN
    INTEGER J,K;
    REAL TEMP;
    EJECT(1);

    OUTIMAGE;
    OUTTEXT(" OVERALL MEAN FOR THE THREE NETWORK PARAMETERS");
    OUTIMAGE;
    OUTTEXT("************************************************");
    OUTIMAGE; OUTIMAGE;
    OUTTEXT(" AVERAGE DELAY");
    OUTTEXT("     THROUGHPUT    NET DELIVERY RATIO");
    OUTIMAGE;
    OUTTEXT("   ");
    OUTFIX(XBAR(1),3,8); OUTTEXT("           ");
    OUTFIX(XBAR(2),3,5); OUTTEXT("             ");
    OUTFIX(XBAR(3),5,8);
    OUTIMAGE;
    OUTIMAGE;OUTIMAGE;

    COMMENT OUTTEXT("XBAR COVARIANCE MATRIX") OUTIMAGE;
    COMMENT FOR J:=1 STEP 1 UNTIL NO_BATCHES DO
    BEGIN
        FOR K:=1 STEP 1 UNTIL NO_BATCHES DO
            OUTREAL(COV(1,J,K),2,8);
    COMMENT OUTIMAGE  OUTIMAGE END; OUTIMAGE; OUTIMAGE;

    OUTTEXT("AUTOCOVARIANCE FUNCTION");
    OUTIMAGE;
    OUTTEXT("**********************");
    OUTIMAGE;
    OUTIMAGE;
    OUTTEXT("                          NETWORK PARAMETERS "); OUTIMAGE;
    OUTTEXT("           AVERAGE DELAY");
    OUTTEXT("     THROUGHPUT    NET DELIVERY RATIO");
    OUTIMAGE;
    OUTIMAGE;

    FOR J:=0 STEP 1 UNTIL NO_BATCHES-1 DO BEGIN
        OUTTEXT(" R("); OUTINT(J,1); OUTTEXT(")");
        COMMENT OUTTEXT("      ");
        OUTREAL(R(0,J)/(NO_BATCHES-J),6,18);
        COMMENT OUTTEXT("          ");
        OUTREAL(R(1,J)/(NO_BATCHES-J),6,18);
        COMMENT OUTTEXT("          ");
        OUTREAL(R(2,J)/(NO_BATCHES-J),6,18);
        OUTIMAGE;
        END;   OUTIMAGE;
END ***STATPRINT***;


COMMENT
COMPUTE THE CORRELATION BETWEEN THE BATCHES FOR THE THREE PERFORMANCE
MEASURES;
PROCEDURE CORELATION(CIDELAY,CITHRU,CINET,R,XBAR);
LONG REAL ARRAY CIDELAY, CITHRU, CINET;
LONG REAL ARRAY R;
LONG REAL ARRAY XBAR;
BEGIN
    INTEGER J;
    LONG REAL ARRAY CORR(0:3,0:NO_BATCHES-1);
    OUTIMAGE; OUTIMAGE;
    OUTTEXT("CORRELATION BETWEEN THE NEIGHBOURING BATCH MEANS");
    OUTIMAGE;
```

```
    OUTTEXT("*************************************************");
    OUTIMAGE; OUTIMAGE;
    OUTTEXT("  RHO                    NETWORK PARAMETERS ");
    OUTIMAGE; OUTTEXT("             AVERAGE DELAY");
    OUTTEXT("      THROUGHPUT      NET DELIVERY RATIO");
    OUTIMAGE;
    OUTIMAGE;

    FOR J:=1 STEP 1 UNTIL NO_BATCHES-1 DO BEGIN
        CORR(0,J):= R(0,J)/R(0,0);
        CORR(1,J):= R(1,J)/R(1,0);
        CORR(2,J):= R(2,J)/R(2,0);
        OUTINT(J,4);
        COMMENT OUTTEXT("           ");
        OUTREAL(CORR(0,J),6,18); COMMENT OUTTEXT("       ");
        OUTREAL(CORR(1,J),6,18); COMMENT OUTTEXT("         ");
        OUTREAL(CORR(2,J),6,18);
        OUTIMAGE;
        END;   OUTIMAGE;
        COMMENT  BATCH_INDE_DELAY(CIDELAY,CORR,R,XBAR);
END  ***CORELATION***;


PROCEDURE BATCH_INDE_DELAY(CIDELAY,CORR,R,XBAR);
LONG REAL ARRAY CIDELAY;
LONG REAL ARRAY CORR,R;
LONG REAL ARRAY XBAR;
BEGIN
COMMENT TESTSTAT CONTAINS THE VALUE USED FOR BATCH INDEPENDENCE TEST;
    LONG REAL  TESTSTAT;

    TESTSTAT := CORR(0,1) +((CIDELAY(1)-XBAR(1))**2 +
                (CIDELAY(NO_BATCHES)-XBAR(1))**2)/
                (2*(NO_BATCHES-1)*(R(1,0)/NO_BATCHES));
                OUTIMAGE;

    OUTIMAGE;
    OUTTEXT("TEST STATISTIC FOR INDEPENDENCE TEST");  OUTIMAGE;
    OUTTEXT("*************************************"); OUTIMAGE;
    OUTIMAGE;
    OUTTEXT("    FOR PARAMETER AVERAGE DELAY"); OUTIMAGE;
    OUTTEXT("    --------------------------");
    OUTIMAGE;
    OUTTEXT("        ");
    OUTREAL(TESTSTAT,6,18);
    OUTIMAGE; OUTIMAGE;
END  ***ESTICORR****;



COMMENT
THIS PROCEDURE COMPUTES THE VARIANCE AND 95% CONFIDENCE INTERVAL FOR THE
THREE PERFORMANCE MEASURES;

PROCEDURE BATCH_CONFINT(XBAR,R);
LONG REAL ARRAY XBAR; LONG REAL ARRAY R;
BEGIN
COMMENT
VAR HOLDS THE VARIANCE OF THE OVERALL MEAN;
HALF_WIDTH HOLDS THE CONFIDENCE INTERVAL HALF LENGTHS;
LOW HOLDS THE LOWER CONFIDENCE LIMIT;
HIGH HOLDS THE UPPER CONFIDENCE LIMIT;

    LONG  REAL  ARRAY VAR(1:3);
    LONG  REAL  ARRAY HALF_WIDTH(1:3);
    LONG  REAL  ARRAY LOW(1:3);
```

```
    LONG  REAL  ARRAY HIGH(1:3);
COMMENT
TVALUE HOLDS T VALUE FOR (NUMBER OF BATCHES-1) DEGREES OF FREEDOM;
    REAL TVALUE;
    INTEGER I, J;



    IF NO_BATCHES=5 THEN TVALUE := 2.776
       ELSE IF NO_BATCHES=10 THEN TVALUE := 2.093
            ELSE IF NO_BATCHES=15 THEN TVALUE := 2.045
                 ELSE IF NO_BATCHES=20 THEN TVALUE := 1.96
                      ELSE IF NO_BATCHES=40 THEN TVALUE := 1.96;

    OUTTEXT("TVALUE      "); OUTFIX(TVALUE,3,5) ; OUTIMAGE;

COMMENT
COMPUTE VARIANCE, HALF_WIDTH, LOWER AND HIGHER CONFIDENCE LIMITS;
    FOR I := 1 STEP 1 UNTIL 3 DO BEGIN
        VAR(I) := R(I-1,0)/(NO_BATCHES*(NO_BATCHES-1));
        HALF_WIDTH(I):= TVALUE*SQRT(VAR(I));
        LOW(I):=XBAR(I)-HALF_WIDTH(I);
        HIGH(I):= XBAR(I)+HALF_WIDTH(I);
        END;

    COMMENT OUTTEXT("S.D.    :")
    OUTREAL(SQRT(VAR),6,18)
    OUTIMAGE;

COMMENT CALL THE PRINT CONFIDENCE LIMITS PROCEDURE;
    PRINT_CONFINT(LOW,HIGH,HALF_WIDTH,VAR);
END ***CONFINT***;


COMMENT PRINT THE VARIANCE, LOWER AND HIGHER C.I. LIMITS;

PROCEDURE PRINT_CONFINT(LOW,HIGH,HALF_WIDTH,VAR);
LONG REAL ARRAY LOW,HIGH,HALF_WIDTH, VAR;
BEGIN
    OUTIMAGE; OUTIMAGE;
    OUTTEXT("0.95 CONFIDENCE INTERVAL ESTIMATES");
    OUTIMAGE;
    OUTTEXT("**************************************");
    OUTIMAGE; OUTIMAGE;
    OUTTEXT("                        NETWORK PARAMETERS ");
    OUTIMAGE; OUTTEXT("              AVERAGE DELAY");
    OUTTEXT("     THROUGHPUT    NET DELIVERY RATIO");
    OUTIMAGE;
    OUTTEXT("LOWER       ");

    FOR I := 1 STEP 1 UNTIL 2 DO BEGIN
        OUTFIX(LOW(I),3,7); OUTTEXT("             ");
        END;
    OUTFIX(LOW(3),6,8);
    OUTIMAGE;
    OUTTEXT("UPPER       ");
    FOR I := 1 STEP 1 UNTIL 2 DO BEGIN
        OUTFIX(HIGH(I),3,7);OUTTEXT("             ");
        END;
    OUTFIX(HIGH(3),6,8);
    OUTIMAGE;
    OUTTEXT("HALF WIDTH  ");
    FOR I := 1 STEP 1 UNTIL 3 DO BEGIN
        OUTREAL(HALF_WIDTH(I),6,12); OUTTEXT("     ");
        END;
    COMMENT OUTFIX(HALF_WIDTH(3),6,8);
    OUTIMAGE;
```

```
    OUTTEXT("VARIANCE    ");
    FOR I := 1 STEP 1 UNTIL 3 DO BEGIN
        OUTREAL(VAR(I),6,12);        OUTTEXT("     ");
        END;
    COMMENT OUTFIX(VAR(3),6,8);
    OUTIMAGE;
    OUTIMAGE;
END   *****PRINTSTAT***;


COMMENT
NO_BATCHES HOLDS THE NUMBER OF BATCHES FOR WHICH THE DATA HAS TO BE
ANALYSED;

    INTEGER NO_BATCHES;
    NO_BATCHES := BATCH_COUNT;
    TITLE(UPDATE.MEAN_DELAY,UPDATE.THROUGHPUT,UPDATE.NET_RATIO);
    XBAR_COV_STAT(UPDATE.MEAN_DELAY,UPDATE.THROUGHPUT,UPDATE.NET_RATIO);
END ***DATANALYSIS***;



COMMENT
NUM_BATCHES = NUMBER OF BATCHES FOR THE ENTIRE SIMULATION RUN (READ IN).
NUM_ENTRIES = NUMBER OF ENTRIES RELATING TO THE NETWORK CONNECTION INFO
              + 1 (COUNT END MARKER -1 ALSO), (READ IN).
NUM_TRANSITIONS = NUMBER OF TRANSITIONS THAT TAKE PLACE DURING A BATCH
                  RUN, INCREMENTED EACH TIME A TRANSITION TAKES PLACE.
BATCH_COUNT = USED TO CONTROL THE FOR LOOP FOR THE PROCESS NEW_BATCH;
   INTEGER NUM_BATCHES,NUM_ENTRIES,NUM_TRANSITIONS, BATCH_COUNT;
COMMENT VARIABLES FOR RANDOM NUMBERS USED IN THE SIMULATION;
   INTEGER U1,U2;
COMMENT INT USED TO ACTIVATE DATA ANALYSIS AFTER 5, 10, 20, 40 BATCHES;
   INTEGER INT;
COMMENT
USED TO SPECIFY THE NUMBER OF OBSERVATIONS REQUIRED FOR STEADY STATE
CONDITIONS AND THE NUMBER OF OBSERVATIONS TO BE COLLECTED ONCE STEADY
STATE IS ACHIEVED. THIS IS ALSO = BATCH SIZE.
   INTEGER OBS_FOR_STEADY_STATE, OBS_IN_STEADY_STATE;
COMMENT
THESE ARE THE DIFFERENT TYPES OF CONTROL PACKETS EXCEPT 'DATA' WHICH
SPECIFIES A MESSAGE PACKET;
   INTEGER REQUEST,DATA,CHANGE,ACK,OVER,VOTE;


COMMENT
NUMBER OF NODES IN THE NETWORK, NUMBER OF APEX CANDIDATES, NUMBER OF
CONFIGURATIONS (NUMBER OF TRAFFIC MATRICES - 1), (ALL READ IN);
   INTEGER NO_NODES,NO_APEXES,NO_CONFIG;

COMMENT
APEX NODE, SEQUENCE NUMBER OF THE PACKET TO BE GENERATED NEXT,
NUMBER OF PACKETS RECEIVED AT THEIR FINAL DESTINATION;
   INTEGER APEX,SEQNR,RECEIVED;

   INTEGER I,L;

COMMENT APEX TRANSITION TIME IN MILLISECONDS;
   REAL TRANSIT_TIME;

COMMENT
   TOTAL DELAY EXPERIENCED BY ALL PACKETS RECEIVED AT THEIR FINAL
   DESTINATION IN MILLISECONDS, AVERAGE DELAY FOR A BATCH, THROUGHPUT FOR
   A BATCH, NETWORK DELIVERY RATIO FOR A BATCH;
   LONG REAL TOTAL_DELAY, AVG_DELAY,THRU_PUT1,THRU_PUT2;
COMMENT SIMULATION START TIME OF THIS NETWORK;
```

```
    LONG REAL START_TIME;

COMMENT
POINTER TO THE NETWORK DATA STRUCTURE.
POINTER TO THE DATA STORAGE THAT HOLDS THE INPUT DATA.
POINTER TO THE ACCOUNTING CLASS THAT HOLDS THE VARIOUS PERFORMANCE ARRAYS
POINTER TO THE CLASS DATA ANALYSIS THAT ANALYSES THE OUTPUT DATA AFTER
              SPECIFIED BATCHES HAVE COMPLETED;
    REF (NETWORK) NET;
    REF (DATA_STORAGE) DATA_STORE;
    REF (ACCOUNTING) UPDATE;
    REF (DATA_ANALYSIS) STATISTICS;

COMMENT INTERVAL OF STATISTICS COLLECTION IN MILLISECONDS;
    REAL SLEEP_TIME;

COMMENT FLAG FOR APEX TRANSITION AND STEADY STATE;
    BOOLEAN TRANSITION, STEADY_STATE;

COMMENT INITIALIZE BOOLEAN STEADY STATE TO FALSE AT START OF SIMULATION;
    STEADY_STATE:= FALSE;


    DATA    :=0;   COMMENT DEFINITION OF A DATA PACKET;
    REQUEST :=1;   COMMENT DEFINITION OF 'TR' (TRANSITION REQUEST);
    VOTE    :=2;   COMMENT DEFINITION OF 'TV' (TRANSITION VOTE);
    ACK     :=3;   COMMENT DEFINITION OF 'TA' (TRANSITION ACKNOWLEDGE);
    CHANGE  :=4;   COMMENT DEFINITION OF 'TC' (TRANSITION COMMIT);
    OVER    :=5;   COMMENT DEFINITION OF 'TO' (TRANSITION OVER);

COMMENT RANDOM NUMBERS AND INT INITIALIZATION;
    U1      := 984366;
    U2      := 36677;
    INT     := 1;

COMMENT
INITIALIZATIONS STEADY STATE STARTS AFTER 45000 PACKETS ARE RECEIVED AT
THEIR FINAL DESTINATION;
    OBS_FOR_STEADY_STATE   := 45000;

COMMENT BATCH SIZE = 30000;
    OBS_IN_STEADY_STATE    := 30000;

COMMENT
INITIALIZATIONS FOR DAEMON STATISTICS COLLECTOR AND RECONFIGURATION
LENGTH IN MILLISECONDS;
    SLEEP_TIME   := 500.0;
    TRANSIT_TIME := 300.0;

READ IN THE FOLLOWING, THEY HAVE BEEN DESCRIBED IN THE TYPE DECLARATIONS;
    NUM_BATCHES := ININT;
    NO_CONFIG   :=ININT;
    NUM_ENTRIES := ININT;
    NO_NODES    := ININT;
    NO_APEXES   := ININT;

COMMENT CREATE AN INSTANCE OF THE ACCOUNTING CLASS AND DATA STORAGE;
    UPDATE      :- NEW ACCOUNTING;
    DATA_STORE  :- NEW DATA_STORAGE;
    OUTIMAGE;

COMMENT START PROGRAM OUTPUT;
    OUTLINE("PROGRAM OUTPUT BEGINS FOR THE DHLAN");
    OUTLINE("USING METHOD OF BATCH MEANS FOR DATA COLLECTION");
    OUTIMAGE;
       OUTLINE("NETWORK PARAMETERS:");
       OUTI("    NO OF NODES         :", NO_NODES);
```

```
      OUTI("    NO OF APEXES        :", NO_APEXES);

      OUTTEXT("    TRANSITION TIME     :");
        OUTFIX(TRANSIT_TIME,3,10); OUTTEXT(" MILLISECONDS");OUTIMAGE;
      OUTTEXT("FIRST U1 :"); OUTINT(U1,20); OUTIMAGE;
      OUTTEXT("FIRST U2 :"); OUTINT(U2,20); OUTIMAGE;

COMMENT
CREATE A NETWORK AND WAIT FOR THE SPECIFIED NUMBER OF BATCHES TO BE
COMPLETED BEFORE ACTIVATION;
      NET :- NEW NETWORK(DATA_STORE);
      PASSIVATE;

END;
/*
//GO.SYSIN DD *
      10 3 21
      11 4
  1  2
  1  3
  1  4
  2  5
  2  6
  3  7
  3  8
  4  9
  6  10
  6  11
  -1
  20.00  0.000   0  15  15  15  10   5  10  10  10   5   5
  19.70  0.880  15   0  10  10  15  15   5   5   5  10  10
  17.02  0.760  15  10   0  10  10   5  15  15  10   5   5
  14.33  0.640  15  10  10   0  10  10  10  10  20   2   3
   8.96  0.400  10  15  10  10   0  10  10  10  10   8   7
  15.22  0.680   5  15   5  10  10   0  10  10   5  15  15
   8.96  0.400  10   5  15  10  10  10   0  20  10   5   5
   8.96  0.400  10   5  15  10  10  10  20   0  10   5   5
   8.96  0.400  10   5  10  20  10   5  10  10   0  10  10
   8.96  0.400   5  10   5   2   8  15   5   5  10   0  35
   8.96  0.400   5  10   5   3   7  15   5   5  10  35   0

6900.0 5 2
  14.53  0.438   0  30  25  25   1   2   5   5   5   1   1
  15.00  0.000  25   0   5   5  25  25   3   1   1   5   5
  14.20  0.428  25  15   0   5   1   1  25  25   1   1   1
  12.41  0.364  30  20  10   0   2   2   2   2  30   1   1
  10.48  0.316  15  50   1   1   0  20   1   1   1   5   5
  15.00  0.452   5  35   1   1   5   0   1   1   1  25  25
  11.01  0.332  20  10  35   5   1   1   0  25   1   1   1
  10.88  0.328  20  10  35   5   1   1  25   0   1   1   1
   9.49  0.286  20  25  20  45   2   2   2   2   0   1   1
  11.01  0.332   5  25   1   1  10  30   1   1   1   0  25
  11.01  0.332   5  25   1   1  10  30   1   1   1  25   0

7000.0 7 3
  29.02  1.336   0  10  35  10   1   1  20  20   1   1   1
  20.07  0.924  10   0  30   5   5   5  20  20   1   2   2
  30.00  0.000  20  15   0  15   5   5  15  15   5   3   2
  12.00  0.552  15  10  24   0   2   2  15  15  15   1   1
   5.87  0.270  10  15  24   5   0   5  15  15   1   5   5
  12.51  0.576   5  10  20  10   4   0  15  15   1  10  10
  11.30  0.520  25  15  20  15   3   3   0  10   3   3   3
  11.30  0.520  25  15  20  15   3   3  10   0   3   3   3
   5.91  0.272  10   5  25  15   2   1  20  20   0   1   1
   6.04  0.278   5  10  15   5   5  17  15  15   3   0  10
   6.00  0.276   5  10  15   5   5  17  15  15   3  10   0
```

```
7000.0 9 4
  27.83  1.276   0  10  10  50   5   5   5   5   5   2   3
  22.25  1.020  10   0  10  50   5   5   2   2  10   3   3
  16.10  0.738  10   5   0  50   5   5  10  10   3   1   1
  28.00  0.000  15  15  15   0  10  10  10  10  10   3   2
   6.94  0.318   5  10   5  50   0  10   5   5   5   2   3
  15.53  0.712   5  13   5  50  10   0   1   1   5   5   5
   6.63  0.304   5   5  10  50   5   5   0  10   5   3   2
   6.63  0.304   5   5  10  50   5   5  10   0   5   2   3
   6.81  0.316  10  10  10  50   4   4   4   4   0   2   2
   5.58  0.256   5  10   5  50   5  10   3   2   5   0   5
   5.63  0.258   5  10   5  50   5  10   2   3   5   5   0
/*
```

The vita has been removed from the scanned document