1991

A testbed for distributed query optimization.

Steven G. Popovich
University of Windsor

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

A TESTBED FOR DISTRIBUTED QUERY OPTIMIZATION

by

STEVEN G. POPOVICH

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the
University of Windsor

Windsor, Ontario, Canada
1991

ISBN 0-315-72805-1

Abstract

The thesis explains that it is possible to develop a testbed which will allow users to: 1) define a distributed database environment, 2) define the topology of the network on which the Distributed Database System is stored, 3) define traffic and other dynamic characteristics of the network at a given point in time, and 4) experiment with different heuristics for query optimization.

The purpose of the testbed is to allow users to experiment with different distributed query optimization heuristics which takes into account network traffic and network topology.

The work done to investigate the thesis involved the design, development, porting and installation of two testbeds: a Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN) simulation model using the NETWORK II.5 simulation package and a Hypercube simulation model using the SIMSCRIPT II.5 simulation language.

The first testbed (DQDB) had limited success and therefore did not support the thesis. However, it did prove very successful as a tool for studying MAN Performance. This thesis report documents the independent development of the DQDB MAC Protocol logic needed to build the DQDB simulation model. The report describes the model and presents the results of the investigation into different DQDB MAN architectures.

The second testbed (the Hypercube model) did support the thesis. This thesis report describes the Hypercube topology, the simulation model and the assumptions that the model is based upon. The report also presents results of the investigation into executing different distributed query optimization schedules in the Hypercube simulation.

This thesis report also provides a literature review of Distributed Query Processing and Optimization on Data Communication Networks for students continuing this work.

To my wife Heather,
mother Madelyn
and sister Maricarol.

Acknowledgements

R. Servantis once said, "The journey is better than the inn". I have found this to be true in the pursuit of my Master's Degree and with the writing of this Thesis report. I would like to express my gratitude to all my teachers, fellow students and friends who have made this "journey" such a positive and rewarding experience for me.

I am grateful to Dr. Arunabha Sen and Pradip Maitra, of the Arizona State University, who provided me with the Supercube model and the background knowledge and confidence to undertake this study.

I am grateful to Dr. Mansoor Alam, now with the University of Toledo, who provided me with the help, guidance and support to achieve what I once thought was unattainable; to submit papers to three conferences.

I want to thank Mr. David Gildner, of Dow Chemical Canada Inc. for organizing my Dow seminars, for all of his helpful editorial comments and suggestions on my papers and this Thesis, and for his endless patience in helping me improve my communication skills.

I want to express my gratitude to Dr. Richard Frost for accepting me into the Master's program, for his guidance through the Master's program and for providing an environment in which research and discovery was both possible and enjoyable.

I am deeply indebted to Dr. Subir Bandyopadhyay for providing me with a Research Assistantship, sending me to Arizona State University, for transmitting to me his never ending enthusiasm for learning, and for all the time and effort he has spent over the last two years in helping me develop into a successful Master's candidate.

Finally, I would like to thank my wife Heather for her love, patience and support in helping make a dream come true.

I have travelled through learning. Thank you all so very much.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

THE ADVANCEMENT OF DISTRIBUTED SYSTEMS

Imagine a project involving hundreds of scientists and engineers accessing and contributing to an international distributed database system that grows at more than a terabyte per day. Helen M. Wood[1] predicts that this may become reality over the next decade due to the international effort to improve our understanding of the planet Earth and the role of man in changing Earth's environment. This is just one example of a complex application that will challenge information technology in the 1990s and beyond.

Advances in VLSI and communications technology combined with scientific and marketing pressure have made possible the vision of loosely coupled, distributed heterogeneous systems supporting users from many backgrounds [1]; Distributed Database Systems (DDS) collecting, maintaining, and supplying data and answers for many of today's most pressing problems, says Wood.

In the past, most research on DDSs centered on special purpose distributed database systems [2]. It is predicted that in the future much of the research and development will be on general purpose DDSs [1], [3], [4].

---

[1] Computer Society President's Message, IEEE Computer Magazine, March 1990.

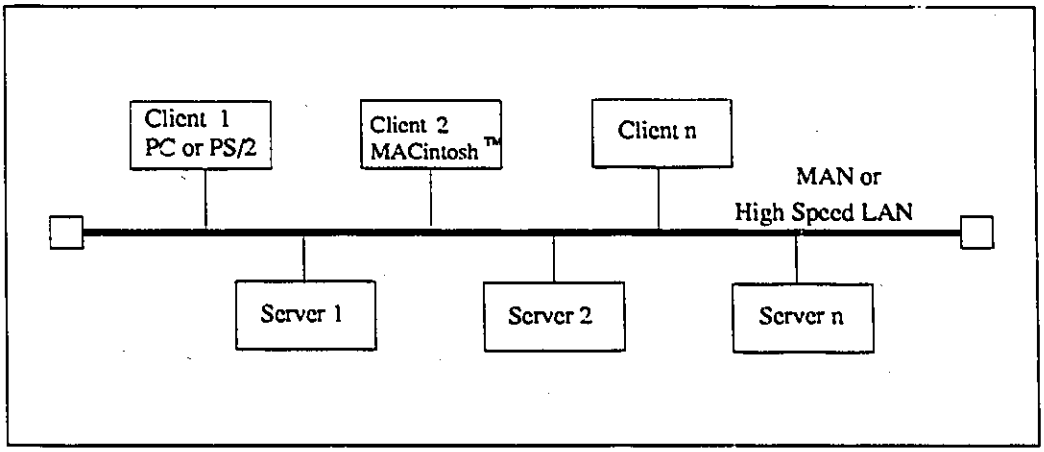


Figure 1 Client-Server Architecture. [5].

Today, research is aimed at developing efficient query optimizers which take into account communication costs, network load, network topology, disk I/O and processing costs, while also evaluating several ways of transporting data across the network to satisfy the requirements of a multi-site transaction.

Another important future development is likely to occur in the area of network system architectures. According to [5], a school of thought has recently developed which advocates that the most attractive model on which to base a Distributed Database System on is the client-server model (shown schematically in Figure 1).

The client-server model provides the immediacy of desktop power together with the host's speed and centralized databases. Since this volume of usage can well exceed the capacity of a single machine or CPU cluster, we will probably see a migration towards specialized servers (i.e. database machines) on MANs and LANs in the future.

The impact of optical fibers on the field of modern public telecommunication

networks and distributed systems cannot be over-emphasized [6], [7]. Fiber optics offer huge bandwidth potential, noise immunity and excellent security. Wavelength Division Multiplexing (WDM) of the lightwave spectrum has made it possible to logically connect computers together with a variety of topologies. Two of these topologies, the Dual Bus and Hypercube, are known for their simplicity, symmetry and ease of node accessibility. It is predicted that networks using these types of architectures will have a tremendous impact on data communications as we move towards ISDN and global high speed networks [8].

How will these gigantic distributed database systems be designed and implemented in order to supply users with timely and reliable information they need? How will Distributed Database Systems (DDS) optimize and process complex queries on such large distributed databases? What public communications systems will be required to handle this amount of data? These are questions that some of today's computer scientists are struggling to answer.

1.2 THE PROBLEM TO BE INVESTIGATED

In [9] and [10] a DDS is defined as a collection of nodes or sites (computers), each of which contains a local database system. Each local database is logically connected together to make up a global database. Each site is able to process local transactions and may participate in the execution of global transactions. The execution of global transactions requires communication among the sites. In [5], 12 rules (Table 1) are discussed which define a fully DDS. The state of DDSs is such that at the present time, no system meets all of these conditions.

1. Local sites should be able to act autonomously.
2. There should be no reliance on a central site.
3. Continuous operation should be possible.
4. Data should be accessible without the user needing to know its location (data transparency).
5. Data fragmentation should be possible, i.e. it should be possible to divide up a given relation into fragments for physical storage purposes (this is only relevant to relational DDS).
6. It should be possible to store distinct copies of a given "fragment" of a relation at many distinct sites (this is only relevant to relational DDS) (data replication).
7. ***Distributed queries should be optimized from a global perspective.***
8. Distributed transaction management must allow concurrency control and deadlock detection.
9. The DataBase Management System (DBMS) should be able to run on different hardware systems.
10. The DBMS should be able to run under different operating systems.
11. The DBMS should support a variety of different communication networks.
12. DBMS independence should be provided, i.e. it should be possible to mix different DBMSs.

Table 1 Distributed Database Rules. [5].

In the past, distributed query optimization (Table 1 rule 7) was carried out with knowledge about:

- the allocation of relations/fragments.
- the replication of relations/fragments.
- the characteristics of the relations involved in the query.
- the nature of the query.
- individual processor load.

An open question is then: "Can Distributed Query Optimization be improved by taking into account the network topology and/or traffic information on individual communication links (network load) in the network?"

To the authors of this document, it seems likely that knowledge of network topology and network traffic will aid in the optimization of distributed queries.

From this discussion, it follows that the development of simulation software which allows a user to take the above factors into account when studying different heuristics for optimizing distributed queries is very useful.

1.3 THE THESIS STATEMENT

The thesis explored in this report is that available simulation tools make it possible to develop a testbed which will allow users to:

- define a distributed database environment
- define a topology of the network on which the DDS is stored
- define traffic and other dynamic characteristics of the network at a given point in time
- experiment with different heuristics for query optimization.

1.4 PURPOSE OF THIS INVESTIGATION

The purpose of this thesis work is to develop a testbed, written with the aid of simulation software, which will enable users to experiment with different Distributed

Query Optimization Heuristics which, along with the characteristics of relations and queries, have the option of taking the network load and network topology into account.

1.5 PREVIOUS WORK RELATED TO THE THESIS

Dr. S. Bandyopadhyay's previous work in this area includes Distributed Query Optimization in a circuit switched Hypercube network [11], [12]. Optimization of a single query was carried out with knowledge of the Hypercube network topology. However, the study did not take into account network load or network traffic.

This thesis work provides an environment for the continuation of this work by providing a testbed for Distributed Query Optimization in packet switched public communication networks.

When it was discovered that the NETWORK II.5 simulation package and the DQDB testbed (first testbed — see Chapters 3 and 4) would not be flexible enough to meet the objectives of the Thesis (see section 6.2), an alternate package and simulation model was sought. The SIMSCRIPT II.5 simulation language was acquired and a study began to see if it would be flexible enough to meet the Thesis objective.

At the same time, P. Maitra [13] was developing a simulation model to study the dynamic characteristics of the Supercube Network [14], the Manhattan Street Network [15] and the Shuffle Exchange Network [16]. The Supercube simulation software was written in SIMSCRIPT II.5 for the VAX/UNIX environment at the Arizona State University.

The Supercube simulation software developed by P. Maitra was made available by Dr. A Sen of the Arizona State University. However, the simulation software needed to be ported and modified to simulate a 32 node Hypercube network (second testbed — see Chapter 5) to run in the SUN/UNIX environment at the University of Windsor.

At this stage, additions to the model are required to verify if the model can execute optimized query schedules. The investigation is documented in Chapter 5.

Other background work relating to the Thesis is discussed in Chapter 2.

1.6 OBJECTIVES AND SCOPE OF THE THESIS WORK

The objectives and scope of the Thesis work are as follows:

1. to explore NETWORK II.5 [17], [18] (discussed in Appendix B) and SIMSCRIPT II.5 [19], [20], [21], [22], [23], [24] (discussed in Appendix C) as tools to simulate computer networks, for the purpose of query optimization.
2. to develop a testbed which models a standard Public Data Communication network (IEEE 802.6 Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN) [25], [26], [27], [28], [29]) using NETWORK II.5 [30].
3. to develop and document detailed DQDB Medium Access Control (MAC) Protocol logic.
4. to develop the following DQDB MAC simulation models:

   - Standard DQDB with Static Boundary and no Bandwidth Balancing — 4, 8, 16 nodes (3 programs).

- Broadcast DQDB with Static Boundary and no Bandwidth Balancing — 16 nodes (one program).
- Standard DQDB with Static Boundary and Uneven Bandwidth Balancing — 4, 8, 16 nodes (3 programs).
- Standard DQDB with Static Boundary and Even Bandwidth Balancing — 4, 8, 16 nodes (3 programs).
- Standard DQDB with Dynamic Boundary and no Bandwidth Balancing — 4, 8, 16 nodes (3 programs).
- Standard DQDB with Dynamic Boundary and Even Bandwidth Balancing — 4, 8, 16 nodes (3 programs).

5. to perform an investigation into the different DQDB MAC MAN architectures to determine:

- Average packet queueing time
- Average slot queueing time
- Average queue length
- Throughput network capacity
- Max operating range
- Slot utilization
- Queueing time by location on network

6. to port and modify the Supercube VAX/UNIX model to the SUN/4 UNIX environment.

7. to develop a testbed which models a proposed Hypercube Public Data Communication Network using SIMSCRIPT II.5.
8. to simulate a 32 node Hypercube network.
9. to add a query processing scheduling subsystem to the model.
10. to use resulting optimized distribution schedules from algorithm SERIAL, PARALLEL, GENERAL and JOIN ([31], [32]) to test the Hypercube testbed.

1.7 ORGANIZATION OF THE THESIS REPORT

Chapter 1 defines the purpose of this investigation and outlines the objectives and scope needed to ensure successful completion of this Thesis project. Chapter 2 surveys previous work in two areas relating to the Thesis. These areas are distributed query optimization and computer networks which facilitate communication in a DDS.

Chapter 3 introduces the first packet switched public communication network investigated (testbed #1): the Distributed Queue Dual Bus (DQDB) Metropolitan Area Network. Chapter 3 overviews the DQDB architecture, describes how the protocol works and discusses how information is transmitted on the communication bus. Chapter 4 documents the independent development of the DQDB MAC Protocol logic and outlines the basic assumptions used in building the DQDB simulation model. Chapter 4 also presents the results of the investigation into different DQDB MAN architectures.

Owing to the fact that the first testbed did not support the Thesis, a second testbed was developed. The second testbed, the Hypercube simulation, is discussed in Chapter 5 along with the basic assumptions used in developing the model. Chapter 5 also reports that it is

possible to develop a testbed which will allow users to: 1) define a distributed database environment, 2) define the topology of the network on which the Distributed Database System is stored, 3) define traffic and other dynamic characteristics of the network at a give point in time, and 4) experiment with different heuristics for query optimization.

Chapter 6 presents a summary of findings and conclusions. Chapter 6 also discusses recommendations and suggests possible future work.

Appendix A and B overviews the NETWORK II.5 and SIMSCRIPT II.5 simulation languages which were used in creating the two testbeds.

CHAPTER 2

BACKGROUND: REVIEW OF THE LITERATURE

DISTRIBUTED QUERY PROCESSING AND OPTIMIZATION

An Overview of a Distributed Query Processing System

The software which manages the DDS is called a Distributed Database Management System (DDMS). Discussed in [33] and [10] are two major components of a DDMS. The first component is called the *user processor* and is responsible for interpreting user input and formatting output, determining the execution strategy (query optimizer), and for monitoring global transaction execution. The second component is called the *data processor* and its responsibility is to optimize resulting local queries, ensure integrity, control concurrency, and to provide recovery should a failure occur. *The acceptance and widespread usage of DDSs will largely depend on how efficient the query optimizer can be made to execute. This is a major area which requires further research.*

In a DDMS, the *data processor* (Figure 2) contains four major components that are directly involved in the execution of a distributed query. They are: the query processing subsystem, the integrity subsystem, the Scheduling Subsystem and the Reliability Sub-

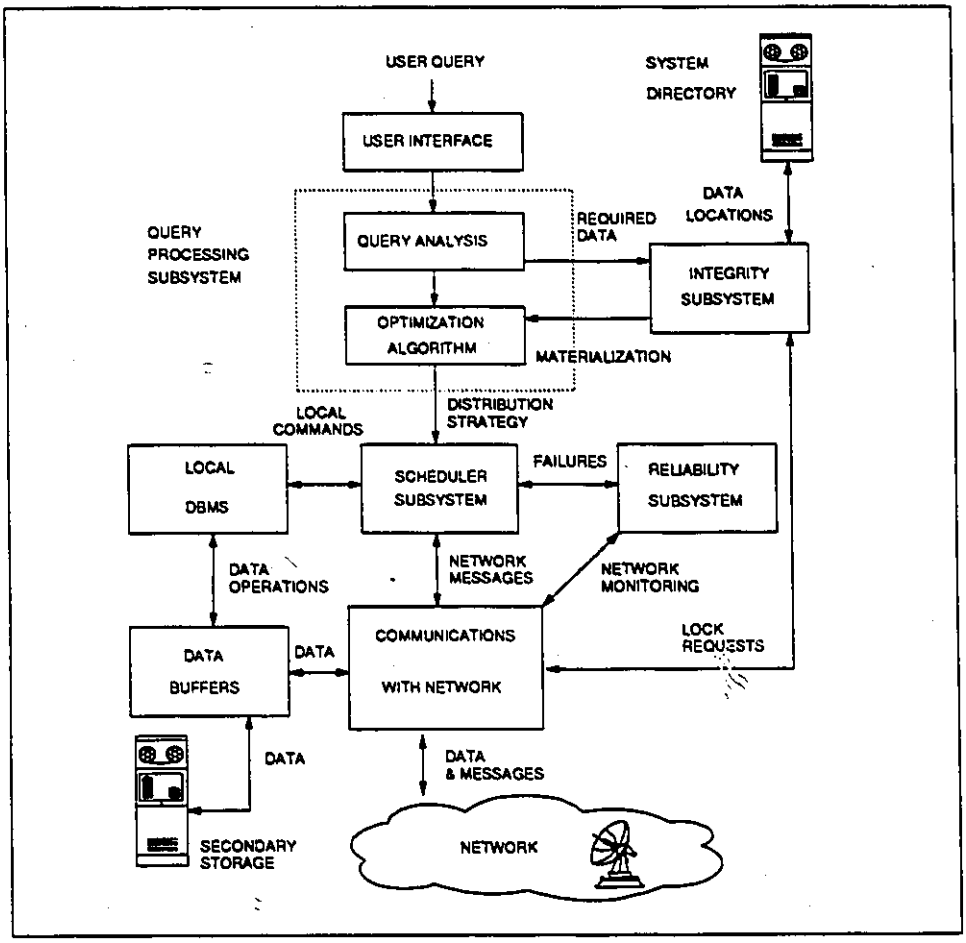


Figure 2 A Distributed Query Processing System. [31]

system. It is these four components and their interaction that provide for efficient and reliable query processing.

Before a query can be optimized, the query processing subsystem must have access to a subset of the database needed to answer the query. This materialization along with the location of the fragments and/or files is provided to the query processing subsystem by the integrity subsystem. In [34], the use of materializations and the role of data

redundancy in query processing is discussed. This approach focuses on how the data available at each site changes as processing proceeds. This information is used in a query optimization algorithm to maximize parallelism and minimize data movement. The integrity subsystem must also solve the problem of controlling concurrent queries and ensure that data integrity and consistency is maintained during updates [35], [36], [37], [38].

Once the query processing subsystem receives a materialization for the query, the optimization algorithm (using algorithm PARALLEL, SERIAL, etc.) can then produce an optimal distribution schedule [39], [40], [41]. The scheduling subsystem coordinates the various schedules in the strategy so that the query response is presented at the result node. In [31] it is stated that a complex distribution strategy will require considerable network coordination of transmission and local processing. *Often simple distribution strategies are* **beneficial** *in a distributed system because the Scheduler is required to transmit fewer messages on the network.*

An objective of a distributed database is to increase the reliability and availability of data in the system. To achieve this objective it is often the case that a number of copies of the same relation are distributed on different nodes in the network [42], [43], [44]. In the case of a node failure there is always another node where a copy of the desired data can be found [45]. The reliability subsystem continuously monitors the system for failures. If a failure occurs the reliability subsystem notifies the scheduling subsystem of the event. The scheduling subsystem either waits for the reliability subsystem to integrate the failed component back into the system or it halts execution and requests that the query

processing subsystem provide a new schedule based on the current status of the system.

What is Query Processing?

The problems involved in distributed query processing are extremely complex. The reason is that there are so many different variations on the kinds of networks available, the way in which these networks are set up, and the way in which data are fragmented, replicated and distributed. In query processing, deciding which processing strategy is the most beneficial requires a great deal of analysis and hence, overhead. Distributed query processing differs from centralized systems in two significant ways :

1. There is a substantial element of processing delay, as the time required to communicate among the sites involved in the query.
2. There is an opportunity for parallel processing since there are several computers involved in handling the query.

Query processing has been an active area of research ever since the beginning of the development of relational databases [46]. Many different query processing strategies can be employed to optimize the processing of a query. A query processing strategy is an equivalence transformation of a query posed in a Data Manipulation Language (DML) [4] to relational algebra [9] which attempts to minimize transmission cost or exploit parallelism. This transformation approach is discussed in [10] and [38]. In this approach, a set of rules is used to transform a query expression into an equivalent expression. The idea is to apply these rules repeatedly to obtain an expression that can be evaluated with a reduced cost.

Distributed Query Processing Strategies

One simple distributed query processing strategy is to ship all relations that are involved in a query to the point where the query originated. This strategy is expensive if large relations have to be transmitted to the originating node and if the result of the query only contains a few tuples and attributes. This strategy is referred to in the literature as the initial feasible solution (IFS) [31].

Another solution is to use the join as a query processing strategy. This involves joining relations before they are transmitted. Computing the join of two relations is expensive. Whenever possible joins should always exploit concurrency. That is, we should carry out joins in parallel on different nodes to save time. It makes no sense to transmit tuples which are not part of the selection criteria. We should "filter" out unwanted information before any transmission is done. This is referred to in the literature as the application of *reducers*. Reducers reduce the amount of information that has to be transmitted. A rule of thumb is to execute unary operations (selection, projection) as soon as possible.

In [47], three parallel join algorithms are developed and implemented on a network of workstations. The three load sharing algorithms are *Simple Sort-Merge Join, Nested Loop Hash Join and Tree Merge Join.* It is claimed that by using these three algorithms and by moving data dynamically to otherwise idle workstations over a LAN, a high degree of parallelism could be achieved through load sharing.

Another strategy is called the Semi-Join strategy. Semi-Join is used in situations where network transmission costs are high. The Semi-Join of relation R, defined over

the set of attributes A, by relation S, defined over a set of attributes B, is the subset of the tuples of R that participate in the join R with S. The advantage of Semi-Join is that it decreases the number of tuples that need to be handled to form the join. It is important in DDSs because it usually reduces the amount of data that needs to be transmitted between sites in order to evaluate a query.

In [48], the Semi-Join strategy is used as the principal reduction operator. [48] defines the Semi-Join operator, explains why Semi-Joins are effective as a reduction operator and presents an algorithm that constructs a cost-effective program of Semi-Joins, given an envelope and a database. [49] also demonstrates the use of the Semi-Join strategy to minimize the intersite data traffic on a point-to-point packet switching communication network.

SEMI-JOINs are beneficial only when a join has good selectivity, in which case the SEMI-JOINs act as a powerful size reducer. [50] states that although the use of Semi-Joins reduces the amount of data transfer and is a valuable tool, it is not always superior to the use of joins. The reasons are:

1. Additional messages may be generated when Semi-Joins are employed.
2. For certain networks, the number of messages exchanged rather than the amount of data transferred may be the dominating factor.
3. Many tactics based on Semi-Join do not take into account local processing cost.
4. Although Semi-Joins can be executed in parallel, the minimization of response time in using Semi-Joins is complicated.

Other techniques used to solve the query processing problem include linear and non-

linear optimization models using large amounts of variables. Some of these variables include the size of relations that are involved in join operations, the range of values which common join attributes have, probabilities about attributes, and the use of knowledge about transmission cost.

How to Measure the Cost of Each Strategy

For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses or CPU processing time. In a distributed system we must take into account the cost of data transmission over the network and the potential gain in performance from having several sites process parts of the query in parallel. A good measure of resource consumption is the total cost that will be incurred in processing a query. *Total cost* is the sum of all times incurred in processing the operations of the query at different sites and intersite communication. Another good measure is the *response time* of a query, which is the time elapsed for executing the query.

The communication cost component is probably the most important factor when considering query optimization in a DDS. Usually the aim of distributed query optimization is simplified to the problem of minimizing communication cost. The communication cost is the time needed for exchanging data between sites participating in the execution of a query. In [31], the cost for answering a query is expressed in terms of the *total cost* measure or the *response time* measure. The data transmission cost between two nodes is defined as a linear function $C(X)= c_0 + c_1X$, where X is the amount of data transmitted. c_0 is defined as the cost to setup a transmission. The total cost measure is computed by adding up all the cost functions between nodes. In [31], the response time measure is

defined as the time from initiation of the query to the time the answer is presented at the query site. Since operation can be executed in parallel at different sites in a DDS, the response time of a query may be significantly less than the total cost.

What is Query Optimization?

Discussed in the previous section were ways of breaking up queries into an equivalence set of operations which attempted to minimize transmission cost and take advantage of parallelism in a distributed system. In the next section, we discuss the optimization of strategies which requires a selection of the most beneficial strategy among alternatives. Query optimization involves improvements in light of the inexact knowledge of the status of the distributed database. The optimization done may not be the best. The optimal strategy may be too difficult to evaluate and could require much more computing to improve on it, which on average may not be dramatically different from the one afforded through a heuristic strategy [39].

Distributed Query Optimization Strategies

Distributed query optimization has received a great deal of attention in the literature. In [51], an algorithm based on the query optimization technique of decomposition is developed. This algorithm is implemented in a distributed INGRES database system. Performance studies of this algorithm and system are reported in [52].

Discussed in [31] are distributed query optimization algorithms using Semi-Join to minimize total time or response time for simple queries (algorithm PARALLEL and SERIAL). Simple queries are defined such that at initial local processing, each relation in

the query contains only one common attribute, which is also the only output of the query. It is claimed that algorithm PARALLEL derives minimal response time $(\hat{r})^2$ distribution strategies by searching for cost beneficial data transmissions in the current system state given by s_i, selectivity p_i and schedule response time r_i of each relation R_i. The selectivity p_i of an attribute is defined as the number of different values occurring in the attribute, divided by the number of all possible values of the attribute. Thus $0 < p_i \leq 1$.

The basic strategy of algorithm PARALLEL is to search for cost beneficial data transmissions by trying to join small relations to large relations. First an Initial Feasible Solution (IFS) is chosen, where all relations are transmitted in parallel to the query site. The algorithm then tries to improve the solution by considering alternative schedules where some relations are sent to an intermediate site. The algorithm does not consider all schedules that could be generated for a given relation R_i. Relations larger than R_i cannot improve the original schedule of R_i and thus are discarded after first ordering the relations by increasing sizes after projection on the join attributes. Algorithm PARALLEL is presented in Table 2.

To minimize *total time* $(\hat{t})$ a serial strategy is discussed in [31]. The outline of the algorithm is described in Table 3. Given an ordering on the required relations of the simple query, the SERIAL strategy consists of transmitting each relation, starting with R_1, to the next relation in a serial order. The strategy is represented by $R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_m \rightarrow R_r$, where R_r is the relation at the result node. There are two cases of the SERIAL strategy. In case 1, R_r is included in its proper order in the transmission pattern,

---

[2] $\hat{r}$ is defined as the $\min(r_i)$ where the minimization ranges over all possible schedules.

1. Order relations R_i such that $s_1 \leq s_2 \ . \ . \ . \leq s_m$.
2. Consider each relation R_i in ascending order of size.
3. For each relation R_j $(j<i)$, construct a schedule to R_i that consists of the parallel transmission of the relation R_j and all schedules of relations R_k $(k<j)$. Select the schedule with minimum response time.

Table 2 Algorithm PARALLEL (Response Time Minimization). [32].

$R_1 - R_2 - \ldots - R_r - \ldots - R_m - \ldots - R_r$. In case 2, R_r is not included in its proper order, $R_1 - R_2 - \ldots - R_m - R_r$. It is claimed that the SERIAL strategy has minimal total time when the relations are ordered so that $s_1 \leq s_2 \leq \ldots \leq s_m$ [31].

Owing to the fact that algorithms PARALLEL and SERIAL only work in specialized situations, the algorithms were extended to work in general query environments [32]. Algorithm GENERAL (discussed in the next section) is a general heuristic that uses an improved exhaustive search to find efficient distribution strategies for general queries.

In [53], query optimization is applied and system performance is presented for a database allocated on a star network.

Owing to the complexity of the distributed query optimization problem, [39] proposes the use of ***heuristic*** algorithms. It is argued that the overhead can be controlled and that the notion that simple greedy heuristic algorithms proposed by many researchers are sufficient in that they are likely to lead to near-optimal strategies. *It is also argued that increasing the overhead in forming optimal strategies is only* **marginally** *beneficial.*

As previously discussed, Hevner and Yao [31] developed two query processing

1. Order relations R_i such that $s_1 \leq s_2 \leq \ldots \leq s_m$.
2. If no relations are at the result node, then select strategy:

   $R_1 - R_2 - \ldots - R_n$ result node. Or else if R_r is a relation at the result node, then there are two strategies:

   $R_1 - R_2 - \ldots - R_r - \ldots - R_n - R_r$

   or

   $R_1 - R_2 - \ldots - R_{r-1} - R_{r+1} - \ldots - R_n - R_r$.
3. Select the one with minimum total time.

Table 3 Algorithm SERIAL (Total Time Minimization). [32].

algorithms called PARALLEL and SERIAL which use Semi-Join as a query processing tactic. These algorithms were designed to minimize response time and total time for a special class of simple queries. They also developed an extended algorithm (Algorithm G) [31] which optimizes general distributed queries. [54] showed that this algorithm had the following drawbacks:

1. The complexity for worst case queries did not have a polynomial bound as stated in [31].
2. Analysis of the quality of the derived processing strategy is difficult.

Algorithm GENERAL

Hevner and Apers recognized these problems and developed an improved algorithm called GENERAL ([32]) to process general queries[3]. The tactic used in the algorithm is to reduce the size of a relation with Semi-Joins on different joining attributes.

[3]A general query means that a relation can contain more than one joining attribute.

Algorithm GENERAL (Table 4) makes the following assumptions:

1. The cost of processing a query is determined by the transmission cost only.
2. All relations are located at different sites.
3. Local processing cost is not taken into account.
4. The query processing strategy is run on a dedicated system.
5. Communication line and subsequent queueing delays are not considered in the algorithm.
6. All initial local processing has been performed.

Each relation R_i is examined in a small to large order (i.e., the index of a relation indicates its relative size after projection on the join attribute): $size(R_1) \leq size(R_2) \leq \ldots\ldots \leq size(R_n)$ to find a schedule that has a minimal response time $\hat{r}_i$ or a minimal total time value $\hat{t}_i$, depending upon the declared cost objective (example given in Figure 3). Each joining domain in the relation R_i is handled separately because of the assumption of domain independence. When the minimal time schedules are found for each join domain, the algorithm integrates these schedules into the overall schedule for relation R_i (Figure 4). The distribution strategy is then constructed by synchronizing the schedules of all required relations in the query. To construct the schedules of the distribution strategy, the algorithm considers only the transmission of join domains between nodes that reduce the size and further transmission cost of the receiving relation by join restrictions (Figure 5) [31].

In [32], it is stated that minimizing response time leads to an increased number of parallel data transmissions in a query processing strategy. To reduce these extra

1. *Do all initial local processing.*
2. Generate candidate relation schedules. Isolate each of the joining attributes, and consider each to define a simple query with an undefined result node.

   a. To minimize *response time*, apply Algorithm PARALLEL to each simple query. Save all candidate schedules for integration in step 3.

   b. To minimize *total time*, apply Algorithm SERIAL to each simple query. This results in one schedule per simple query. From these schedules, the candidate schedules for each joining attribute are extracted. Consider joining attribute d_{ij}. Its candidate schedule is identical to the schedule produced by Algorithm SERIAL, applied to the simple query in which d_{ij} occurs, up to the transmission of d_{ij}. All transmissions after that are deleted from the schedule.

3. *Integrate the candidate schedules.* For each relation R_i, the candidate schedules are integrated to form a processing schedule for R_j. The integration is done by procedure RESPONSE (Table 5) for response time minimization and by procedure TOTAL or procedure COLLECTIVE for total time minimization (Figure 3).
4. *Remove schedule redundancies.* Eliminate relation schedules for relations which have been transmitted in the schedule of another relation.

Table 4 Algorithm GENERAL. [32].

transmissions, algorithm TOTAL is used. By minimizing the total time in a query processing strategy, fewer transmissions will be included and improved actual response times may result in certain systems environments.

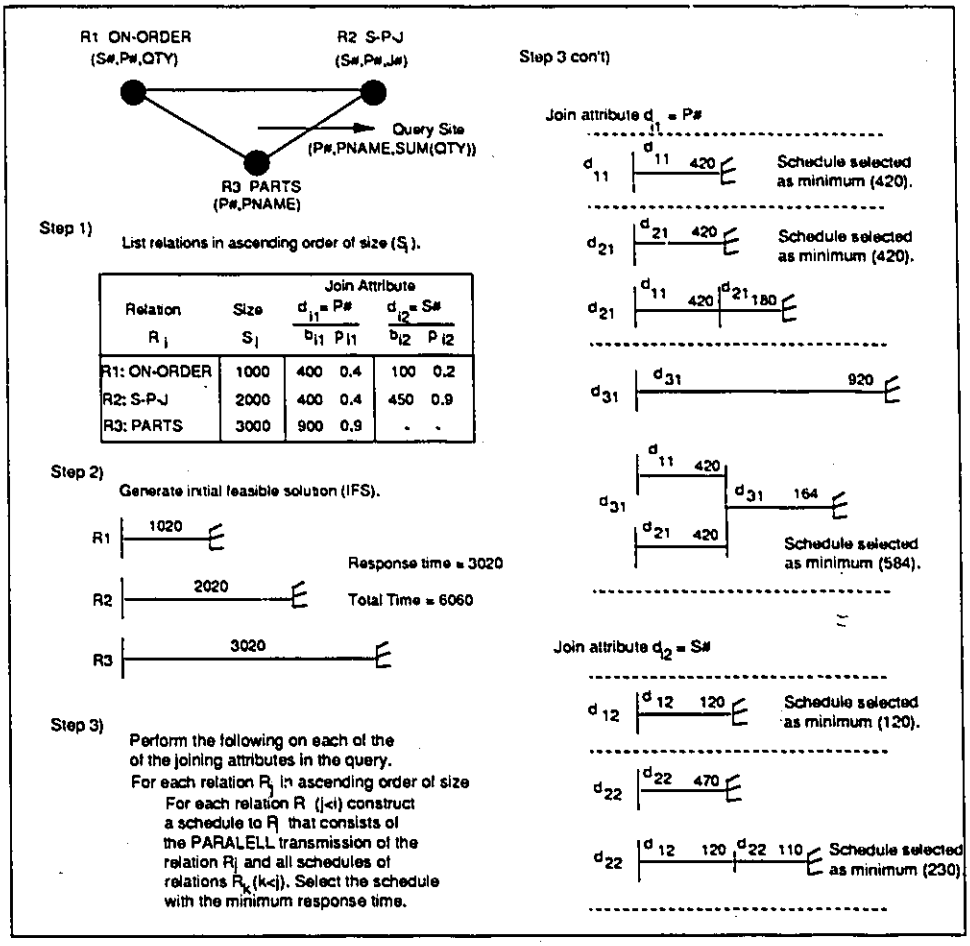


Figure 3 Example of Algorithm GENERAL. [32].

A further reduction in total time is possible if the existence of identical data transmissions in different relation schedules is discovered. A third version of Algorithm GENERAL (COLLECTIVE) is developed to recognize redundant data transmissions among separate relation schedules in the overall query processing strategy. In many cases, the total time of these strategies is less than the total time of strategies produced by Algorithm GENERAL (total time).

1. *Candidate schedule ordering* For each relation R_i order the candidate schedules on joining attribute d_{ij}, $j = 1, 2, ..., \sigma$ in ascending order of arrival time. Let ART_l denote the arrival time of candidate schedule $CSCH_l$. (For the d_{ij} joining attributes not in R_i, disregard the corresponding candidate schedules) (Figure 4).
2. *Schedule integration* For each candidate schedule $CSCH_l$ in ascending order, construct an integrated schedule for R_i that consists of the parallel transmission of $CSCH_l$ and all $CSCH_k$ with $k < l$. Select the integrated schedules with minimum response time (Figure 5).

Table 5 Procedure RESPONSE. [32].

Complexity of Algorithm GENERAL

In [32], algorithm GENERAL is claimed to be an efficient algorithm of polynomial complexity that derives close to optimal query processing strategies for distributed systems. Two versions of Algorithm GENERAL are presented. The first is to minimize response time and the second is to reduce total time.

The major drawback of algorithm GENERAL is its complexity[4]. In [55], a more efficient and less complex method for processing general queries is proposed. An extended version of algorithm PARALLEL and strategy SERIAL is applied to queries to generate transmission schedules. It is claimed that this method gives better and more efficient solutions with a lower order of complexity than Algorithm GENERAL. Work

[4]For each relation (m), for each joining domain (δ), for each incoming domain (m), and for each possible domain candidate (m) the upper bound complexity is $O(m^3)$.

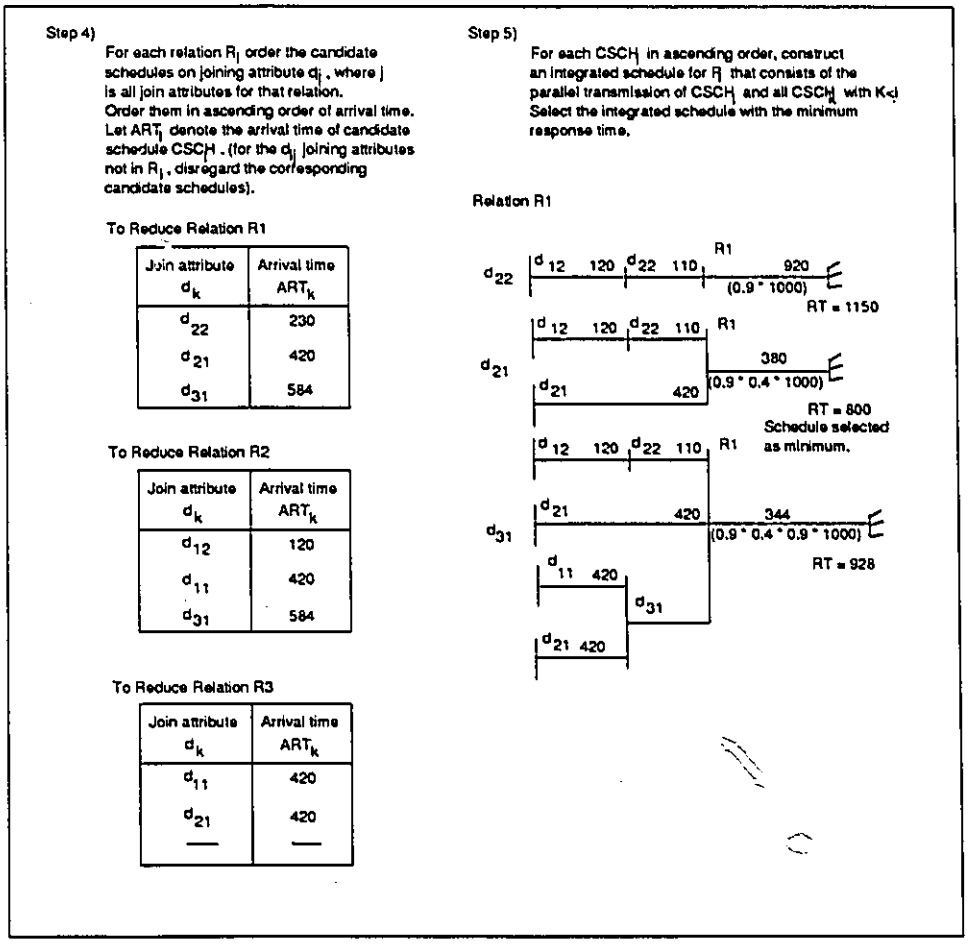


Figure 4 Example of Algorithm GENERAL — Continued.

by [55] defines the solution space of feasible processing strategies for distributed queries. Presented is an optimal, although inherently exponential, optimization algorithm.

Reason for Extending Algorithm GENERAL

The reason why algorithms PARALLEL, SERIAL, and GENERAL are used to test the testbed is that these algorithms can be applied to any distributed query environment. These algorithms assume the network topology to be a wide area point-to-point network.

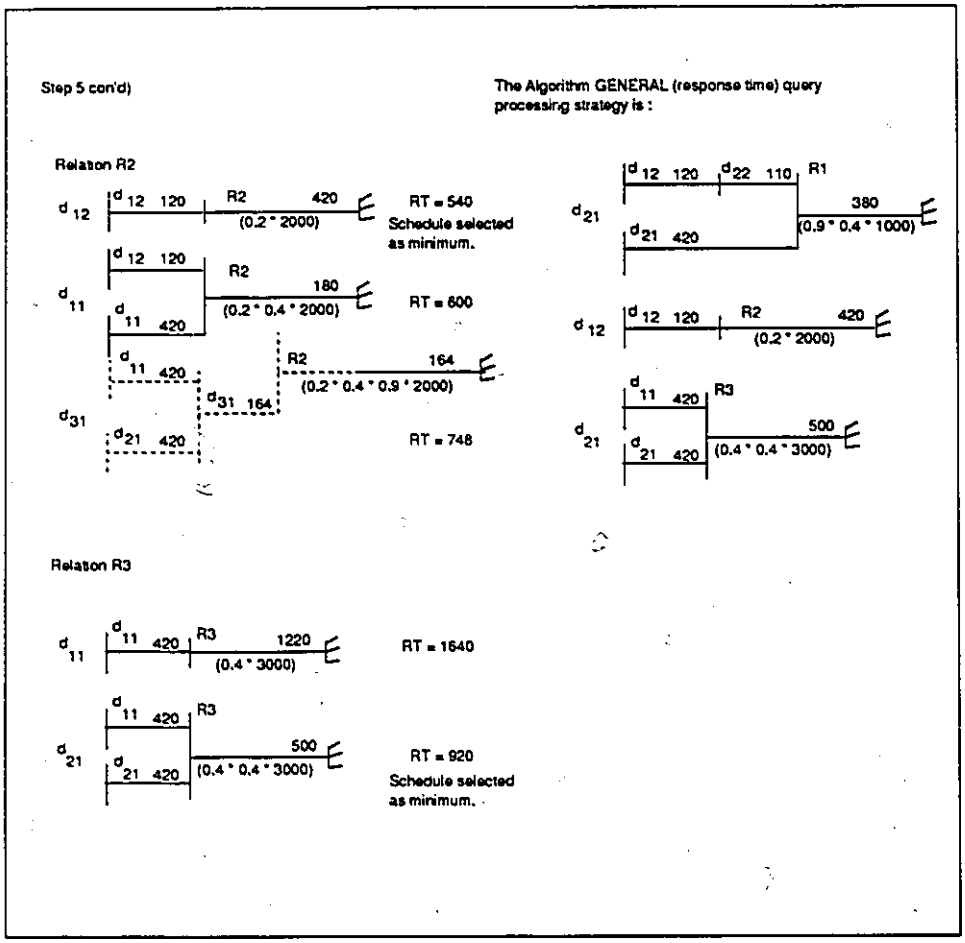


Figure 5 Example of Algorithm GENERAL — Continued.

Also, the resulting optimized schedules are relatively simple to program and can be easily implemented in the Hypercube simulation.

[32] also claims that algorithm GENERAL has the added flexibility that all versions (algorithm GENERAL total time version & algorithm GENERAL response time version) can be implemented together. The optimization objective (total time or response time minimization) can be changed by a simple switch in the program depending upon run-

time factors such as system load or query complexity, An example of this would be to use algorithm GENERAL total time version when network load is moderate to heavy and to use algorithm GENERAL response time version when network load is light.

2.2 PUBLIC DATA COMMUNICATION SYSTEMS WHICH INTERCONNECT DISTRIBUTED DATABASE SYSTEMS

A major problem that distributed systems face, particularly at the national level, is that we do not yet have in place a public communication infrastructure that would allow computers to easily connect to one another over a large geographical area. Although the telephone system is used to that end today, it is inadequate. Speed is low and delays are too long (relative to the speed of computers). The situation resembles the U.S. road system 75 years ago; in principle you could get where you wanted on narrow dirt roads. The massive economic benefits of our current transportation system had to await the modern highway infrastructure.

The Reference Model Used for this Testbed: The OSI Model

Owing to the fact that the Open System Interconnection (OSI) model has achieved universal acceptance, it has allowed communication standards to be developed for each layer of the model. It also provides a reference for discussing communication system design. The network simulation models designed in this document will follow its structure.

The basic idea is to decompose a large problem into several more manageable sub-problems. The model partitions the communication function into layers. Each layer

LAYER	FUNCTION
Physical	Concerned with transmission of unstructured bit streams over a physical medium; deals with the mechanical, electrical, functional, and procedural characteristics to access the physical medium.
Data link	Provides for the reliable transfer of information across the physical link; sends blocks of data (frames) with the necessary synchronization, error control, and flow control.
Network	Provides upper layers with independence from the data transmission and switching technologies used to connect systems; responsible for establishing, maintaining, and terminating connections.
Transport	Provides reliable, transparent transfer of data between end points; provides end-to-end error recovery and flow control.
Session	Provides the control structure for communication between applications; establishes, manages, and terminates connections (sessions) between cooperating applications.
Presentation	Provides independence to the application processes from differences in data representation (syntax).
Application	Provides access to the OSI environment for users and also provides distributed information services.

Table 6 Functions Performed at each Layer of the OSI Model. [56].

performs a subset of the functions required to communicate with another system. Also, each layer is defined so that a change in one layer does not require a change in another. A brief definition of each layer is given in Table 6. A complete definition is given in [56] and [57].

The Physical Transmission of Data

Since the sites in Wide Area Networks (WAN) are distributed physically over a large geographical area, the communication links are likely to be relatively slow and less reliable as compared with Local Area Networks (LANs). Typical WAN links are

telephone lines, microwave links, and satellite channels. In contrast, since all sites in LANs are close together, the communications links are of higher speed and have lower error rates than their counterparts in WANs. The most common links are twisted pair, Baseband coaxial, broadband coaxial, and fiber optics [56], [57].

The sites in a computer communication system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct physical connection between two sites. In most cases it is just not feasible to fully connect all sites together. To solve this problem, other network topologies are used (i.e. ring or bus network) that provide for the sharing of transmission facilities among many sites. This reduces the cost incurred by any pair of sites.

Two of the best known ways to transmit digital information are Wavelength Division Multiplexing (WDM) and Time Division Multiplexing (TDM). In WDM, the frequency spectrum is divided into logical channels, with each user having exclusive possession of this frequency band (frequency sharing). In TDM, each user takes turns accessing the entire bandwidth for short periods of time (round robin). The advantage of multiplexing schemes (as opposed to using a dedicated copper wire between every pair of communicating stations) is that a large number of voice and data transmissions can occur simultaneously, thus making better use of the existing communication facilities.

With the recent developments in light wave technology, it is possible to transmit data on an optical fiber using light pulses to represent bits (a light pulse represents a 1 bit, no light pulse represents a 0 bit). Reasons why fiber optic technology is so important is

that fiber has high bandwidth, is thin and lightweight, is not affected by electromagnetic interference, and has excellent security because it is nearly impossible to wiretap without detection.

One fiber optic transmission standard for MANs, known as Fiber Distributed Data Interface (FDDI), is an example of a token-passing ring capable of 100 Mbits/s operation [58]. FDDI cabling consists of two fiber rings, one transmitting clockwise and the other transmitting counterclockwise. If either one breaks, the other can be used as a backup. To transmit data on the fiber, a station must first capture a token. The station with the token transmits a frame and removes it when it comes around again. The FDDI design specification calls for no more than one error in 2.5×10^{10} bits [59].

Owing to the fact that only one station can transmit at a time, the enormous bandwidth offered by fiber optics is not being utilized. FDDI uses lightwave technology for its superior transmission properties, but the ring does not even come close to tapping the bandwidth of the medium. [60] indicates that concurrency techniques, such as WDM and TDM are the key to tapping this enormous bandwidth. WDM can produce a multitude of non-interfering packets to be simultaneously resident in the network (on one fiber), thereby achieving concurrency by spreading signals over a vast portion of the optical band.

Evolving Computer Networks: Architecture & Importance

In [56], a Local Area Network (LAN) is defined as a communication network that provides interconnection of a variety of data communicating devices within a small area.

LANs support minis, mainframes, terminals and other peripherals. In many cases, these networks can carry not only data but voice, video, and graphics.

The most common type of LAN is the bus or tree using coaxial cable. Rings using twisted pair, coax, or even fiber are an alternative. The data transfer rates on LANs (1 to 20 Mbits/s) are high enough to satisfy most requirements and provide sufficient capacity to permit large numbers of devices to share the network.

The LAN is probably the best choice when a variety of devices and when a mix of traffic types are involved. The LAN, alone or as part of a hybrid local network, has become a common feature of many office buildings and divisions. LANs usually have a length of not more than 2 or 3 miles. This restriction is a result of propagation delay in the network and as we will see is a very important limitation.

Metropolitan Area Networks (MANs) are defined as networks that support two-way communication over a shared medium, such as optical-fiber cable, span a distance of approximately 50 miles, and may offer point-to-point high-speed circuits or packet-switched communication. MANs are expected to transmit data at rates of 150 Mbits/s. They do not, however, have the huge traffic-handling capability of a switched exchange network, such as the present telephone system. Essentially, a MAN is a very large LAN, using access protocols less sensitive to network size than those used in LANs. A cable television (CATV) network, which is essentially a broadcasting system, is not ordinarily classified as a MAN, but can be modified to support two-way MAN service [61].

In [25], it is predicted that high speed Metropolitan Area Networks are ready to challenge the traditional star topology of the voice-only telephone network. It is also

stated that the telephone network is not suited to handle data transmission. LANs and MANs are not scaled-down telephone networks; rather they are scaled-up and serialized computers. These *local* networks stem from computer architecture which is based on the internal computer bus structure where all data flows between central processor, memory, and peripherals. Each one of these units *time-shares* the bus as needed. As a result of this, it would seem like an extremely effective idea to connect computers together via this same type of parallel bus structure. As it turns out, this is not practical. Bus-allocation schemes usually assume very short distances and the parallel cabling would prove to be too expensive and cumbersome. As a result, computers which are attached to LANs share a high-speed serial transmission facility that no single processor dominates.

The problem with LANs is that they have been optimized for distances of up to several miles. MANs on the other hand, have been designed and optimized for longer distances. Also MAN traffic will include both data and voice transmission. As a result the IEEE 802.6 standards committee has attempted to give roughly equal weight to both data and voice requirements. Since digital voice comprises the bulk of the bits sent between locations, any MAN must deal efficiently with it. Typical MAN traffic is expected to include [25]:

- LAN interconnection
- Graphics and digital images
- Bulk data transfer
- Digitized voice
- Compressed video

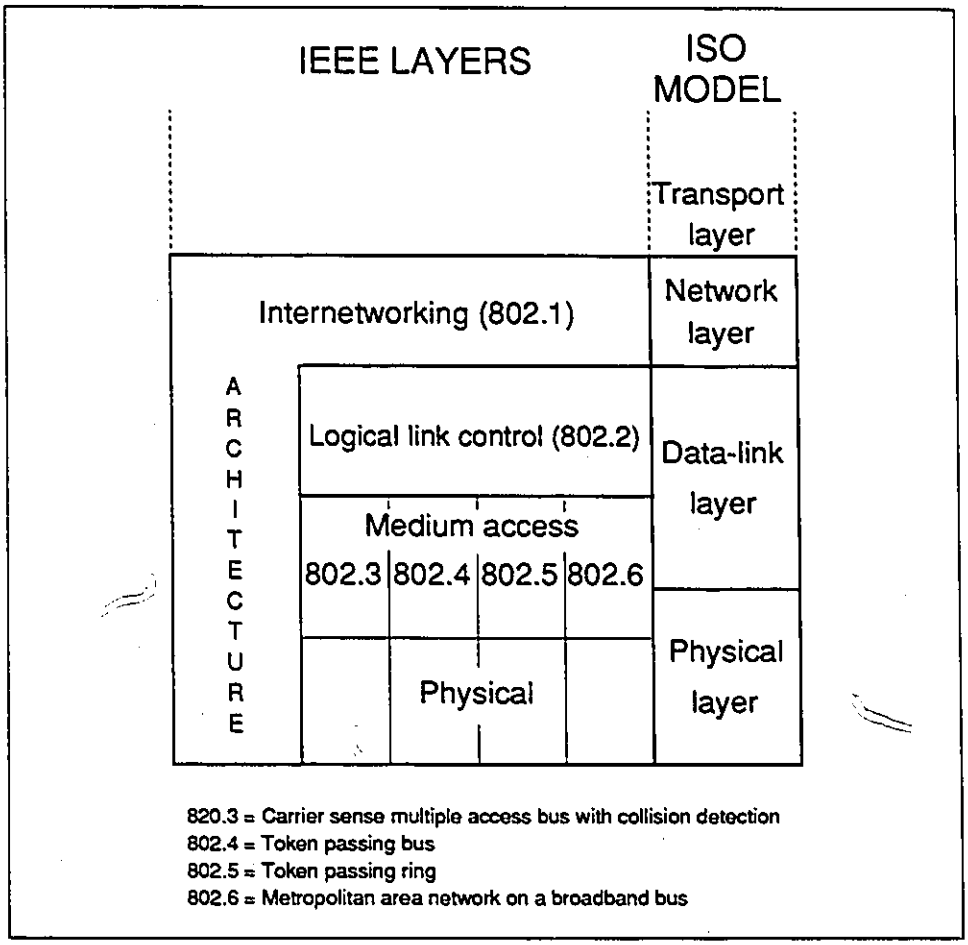


Figure 6 Relationship Between IEEE LAN Family and the ISO Model. [57]

- Conventional terminal traffic

Standards for MANs have been developed as part of the Institute of Electrical and Electronic Engineers (IEEE) 802 LAN project (Figure 6). The MAN group known as the IEEE 802.6 committee is to provide a specification for a city wide network which will carry digital, voice and video as well as interconnect to LANs and other data sources.

LOCAL AREA NETWORKS (LAN)	WIDE AREA NETWORKS (WAN)
Within site up to 3 Miles	Distances up to thousands of miles
High bandwidth (> 1M. bits/s.)	Typical data rates up to 100 k. bits/s.
Simpler protocols	Complex protocols
Interconnect cooperating computers in distributed processing applications	Interconnect autonomous computer systems
Usually operated by the same organization which operates the computers it connects	May be managed by organizations independent of users, e.g. AT&T™
Generally digital signalling over private cables	Often use analogue circuits from the telephone system
Lower error rates (1 in 10^9)	Higher error rates (1 in 10^5)
Can broadcast a single message to multiple destinations	Generally use point-to-point links
Common topologies - bus or ring	Common topologies - mesh or star

Table 7 Characteristics of LANs and WANs. [57].

In [62], a comparison is given between LANs and Wide Area Networks (WANs). Most early networks were WANs designed for voice communication. The arrival of cheap microprocessors has resulted in the proliferation of computers within individual sites. LANs were developed to interconnect these local computers in order to share data and resources. Table 7 compares the characteristics of local and wide area networks.

WANs usually span greater distances, are based on the store-and-forward switching mechanism, and require the routing of packets. LANs on the other hand, span distances up to 3 miles, are based on the ring topology which constitute a single data link, and are usually less error prone. A trend that is emerging is the use of MANs (span 50 miles) to interconnect LANs, and to connect LANs to WANs.

The telephone industry's future standard for its fiber optic WANs is Broadband Integrated Services Digital Network (BISDN). BISDN will provide worldwide uniform digital voice and data services. Compatibility between standard MANs and BISDN is now being worked out. The key here is that the same *header* will be used by both the 802.6 MAN and the BISDN standards. The result of this is that MAN packets will be transmitted on WANs with ease. WANs will then interconnect MANs, and MAN will interconnect LANs. When this happens the *global* LAN will have arrived.

CHAPTER 3

AN OVERVIEW OF THE FIRST NETWORK PROTOCOL INVESTIGATED: THE DISTRIBUTED QUEUE DUAL BUS

The reasons that Distributed Queue Dual Bus (DQDB) has gained such wide-spread acceptance as a public network are that it has outstanding reliability, it is independent of network size and speed, it provides for security, network administration, and bandwidth control, and it has a frame structure that will be compatible with the evolving BISDN. Hence, DQDB can be seen as an evolution path to this future public network technology [27]. It is for this reason that this Medium Access Control protocol (MAC) will be simulated.

3.1 THE DISTRIBUTED QUEUE DUAL BUS ARCHITECTURE

The three main components of the DQDB architecture are the Frame Generating stations (Head Station and Slave Station), two unidirectional buses, and the nodes (computers) which are connected to it. An overview of this architecture is shown in Figure 7. The head station generates the frames on bus A, and the slave station generates the same

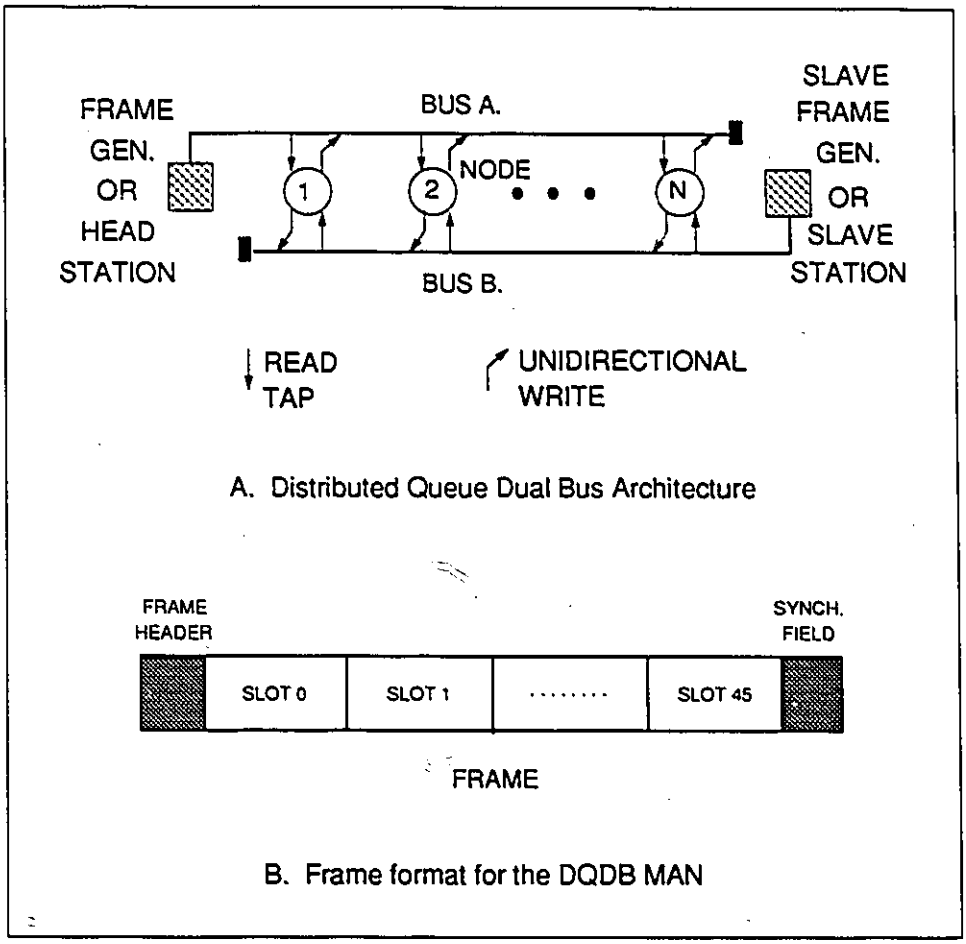


Figure 7 DQDB Bus and Frame Structure. [27].

frame pattern at the same rate on bus B. Each node is connected to each bus via a read and write connection. The read connection is placed ahead of the write connection and allows all data to be copied from the bus unaffected by the node's own writing.

Communication on the DQDB subnetwork is within the frame format shown in Figure 7B. The frame interval is 125 microseconds, matching that of the digital telephony. Frames are subdivided into a fixed number of equal size bits called slots. Slots provide

the subdivision between isochronous (regularly periodic data such as voice and video) and non-isochronous (irregular, bursty data such as file transfers) traffic. A slot may be allocated to carry either type of communication. Slots are allocated for isochronous use based on the proportion of demand for isochronous capacity. The remaining slots are then made available for packet communication.

For isochronous communication, each octet within a slot provides a 64kbit/s channel. By using multiple octets per frame, higher rate circuits can be constructed. For non-isochronous communications, a slot once reserved for access by a node, is used to transfer a single segment of a packet.

3.2 THE DISTRIBUTED QUEUE DUAL BUS PROTOCOL

The protocol uses two control bits, busy and request, to control access to a slot on the forward bus (Figure 8A). An identical, but independent arrangement applies for access to the opposite bus.

When a node has a segment for transmission on the forward bus it will issue a single REQ on the reverse bus. This bit will pass to all upstream nodes, where upstream is defined in relation to the flow on the forward bus. This REQ bit serves as an indicator to the upstream nodes that an additional segment is now queued for access.

Each node keeps track of the number of segments queued downstream from itself by counting the REQ bits as they pass on the reverse bus as shown in Figure 8B. For each REQ passing on the reverse bus, the request (RQ) counter is incremented. One REQ in the RQ counter is canceled each time an empty slot passes on the forward bus.

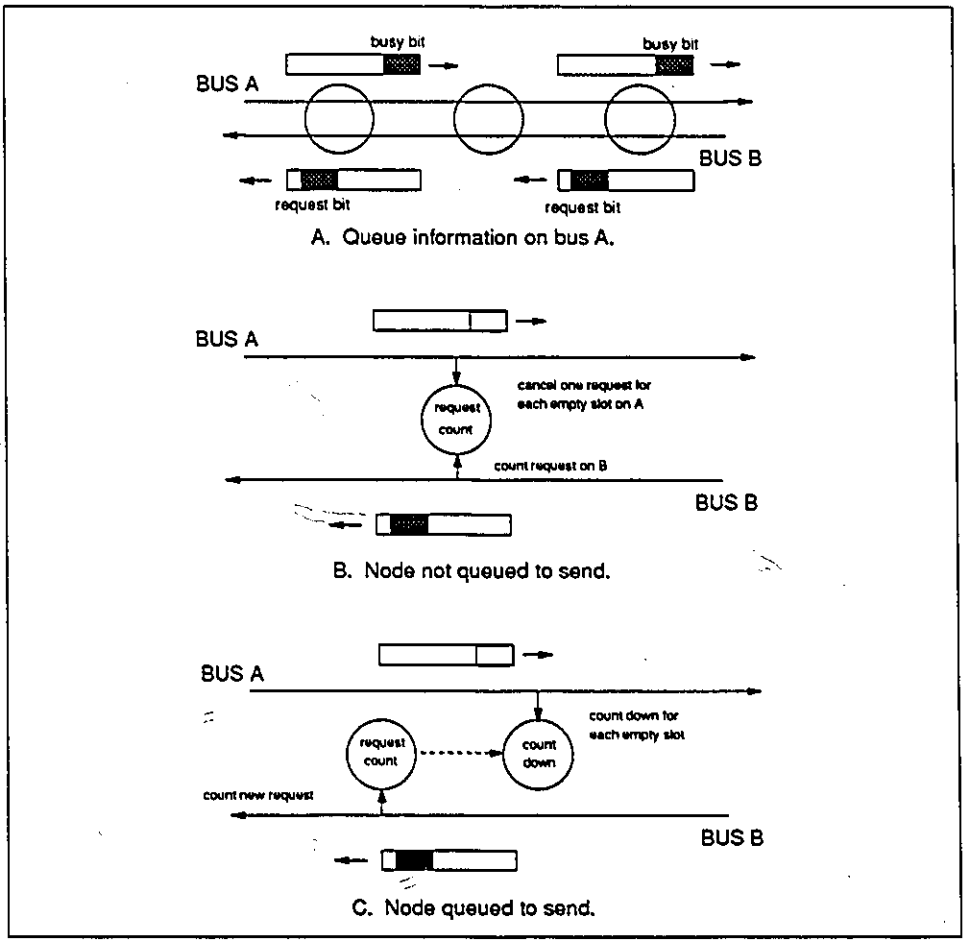


Figure 8 Distributed Queueing. [27].

This is done since the empty segment that passes the node will be used by one of the downstream queued segments. We see that with these two actions the RQ counter keeps a precise record of the number of segments queued downstream.

When a node has a segment for access to the bus, it transfers the current value of the RQ count to its second counter, the countdown (CD) counter (Figure 8C). This action loads the CD counter with the number of downstream segments queued ahead of

it. This, along with the sending of the REQ for the node's segment, effectively places the segment in the distributed queue.

To ensure that the segment registered in the CD counter gains access before the newly queued segment in the given node, the CD counter is decremented for every empty slot that passes on the forward bus. This operation is shown in Figure 8C. The given node can then transmit its segment in the first empty slot after the CD counter reaches zero. The claiming of the first free slot ensures that no downstream segment that queued after the given segment, can push in and access out of order.

During the time that the node is waiting for access for its segment, any new REQs received from the reverse bus are added the RQ counter. Hence, the RQ counter still tracks the number of segments queued downstream and the count will be correct for the next segment access.

Thus, with the use of the two counters in each node, one counting outstanding access requests and the other counting down before access, a first-in-first-out (FIFO) queue is established for access to the forward bus. The queue is established for access to the forward bus. The queue formation is also such that a slot is never wasted on the network if there is a segment queued for it. This is guaranteed since the CD count in the queued nodes represents the number of segments queued ahead. Since at any point in time one segment must have queued first, then at least one node is guaranteed to have a CD count of zero. It is that node that will access the empty slot.

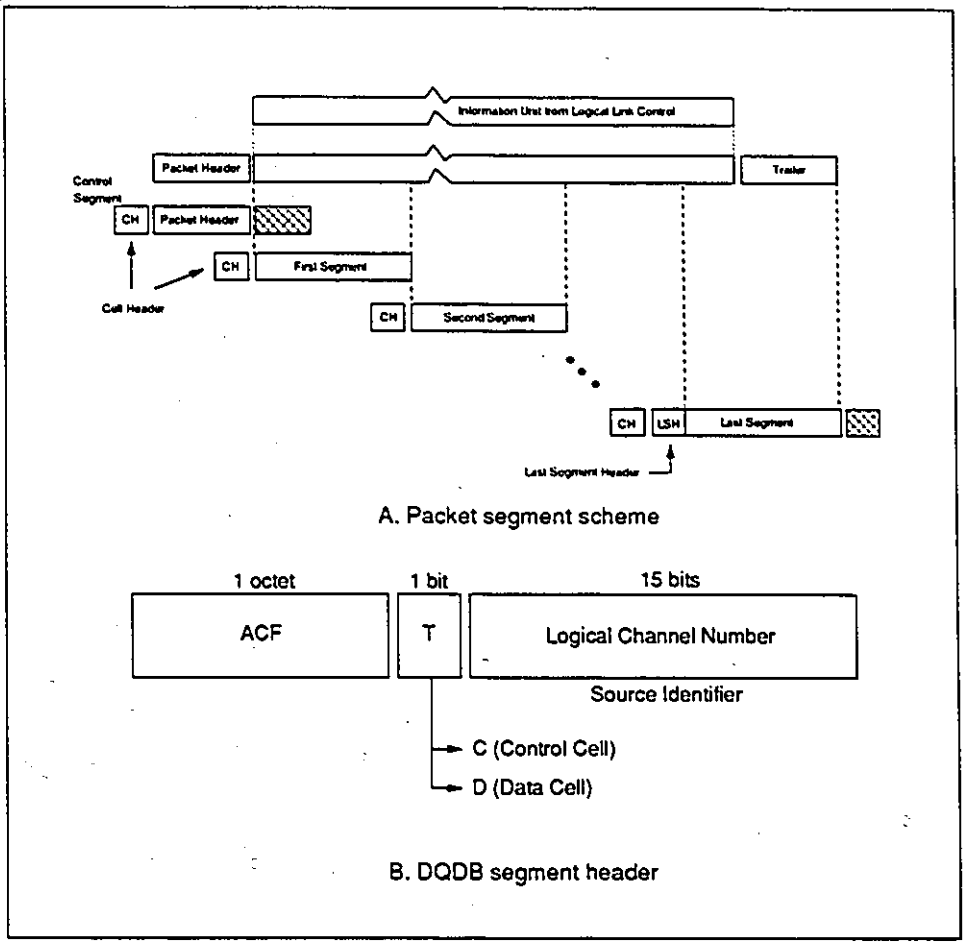


Figure 9 DQDB Segmentation Scheme. [27].

3.3 SEGMENTATION AND REASSEMBLY OF PACKETS

In [27], the scheme used to control the segmentation of packets at the source and the reassembly at the ultimate destination, is described. This process is shown in Figure 9 and involves the fragmentation of the original packet into segments, to match the slot size on the bus.

All segments consist of a header field along with the actual data as shown in Figure

9. The header field consists of two sub-fields. The first is a single bit, the C/D subfield, which is used to indicate whether the segment is a control or a data segment. The second field is the source identifier (SI). This is used to provide the logical linking between slots of the same packet.

The SI is a fifteen bit field. Each node in a DQDB network will have one or more unique SI number(s) related to its physical location along the bus. The SI is in addition to, and independent of, any Network Layer addressing. The allocation of the SI numbers is locally administered and only has significance within a single network. The purpose of the SI is to identify segments of a single packet. This information is used to reassemble the segmented message at the destination. To describe the operation of the packet transfer scheme, the segmentation of the packet at the source is described first and the action of the receiver is considered later.

The train of segments sent by the source is shown in Figure 9. The first segment of a multi-segment message is a control segment. Within this segment a control field immediately after the segment header will indicate the beginning of message (BOM) code. This code signifies the start of a new packet transfer. The SI field of this segment is set to the source's value and the information field of the segment contains additional control information amounting to a packet header. Part of this information is routing information required to transfer the packet to the destination. All routing within the MAN is based on physical access unit addresses, not on the Network Layer addresses. Also, contained within the information field of the first segment is the length of the packet.

All segments of the packet until the last are placed in the information fields of slots

following the first control slot. In the header of each of these slots the C/D bit is reset to indicate data and the SI field is the same value as was sent in the first or control segment. The transfer of the multi-segment packet is completed by sending a further control slot which contains the end of message (EOM) code in the control field after the segment header.

For the transfer of a message that only requires a single segment, the SSM (single segment message) code is used in the control field of the first segment. The SI is not required in this case, however it is still used for consistency in operation.

CHAPTER 4

AN INVESTIGATION OF THE DQDB, THROUGH SIMULATION

SIMULATED DQDB ARCHITECTURE

Distributed queueing is a medium access control protocol (MAC) packet switching scheme that controls the access of fixed length data segments to the slots on the DQDB bus. A current state record is kept in every node which holds the number of segments awaiting access to the bus. When a node has a segment for transmission, it uses its count to determine its position in the distributed queue. If no segments are waiting, access is immediate, otherwise preference is given only to those segments that queued first.

DQDB Frame Structure

Every 125 microseconds the Frame and Slave Frame Generators generate frames for transmission on the forward and reverse buses respectively. The frame size is 19440 bits (CCITT G.707–9, 155,520,000 bits/s / 8000 frames/s = 19440 bits/frame), see Figure 10. Each frame is divided into equal size slots capable of carrying both isochronous and non-isochronous data. Access to isochronous slots is *Pre-Arbitrated* (PA), while access

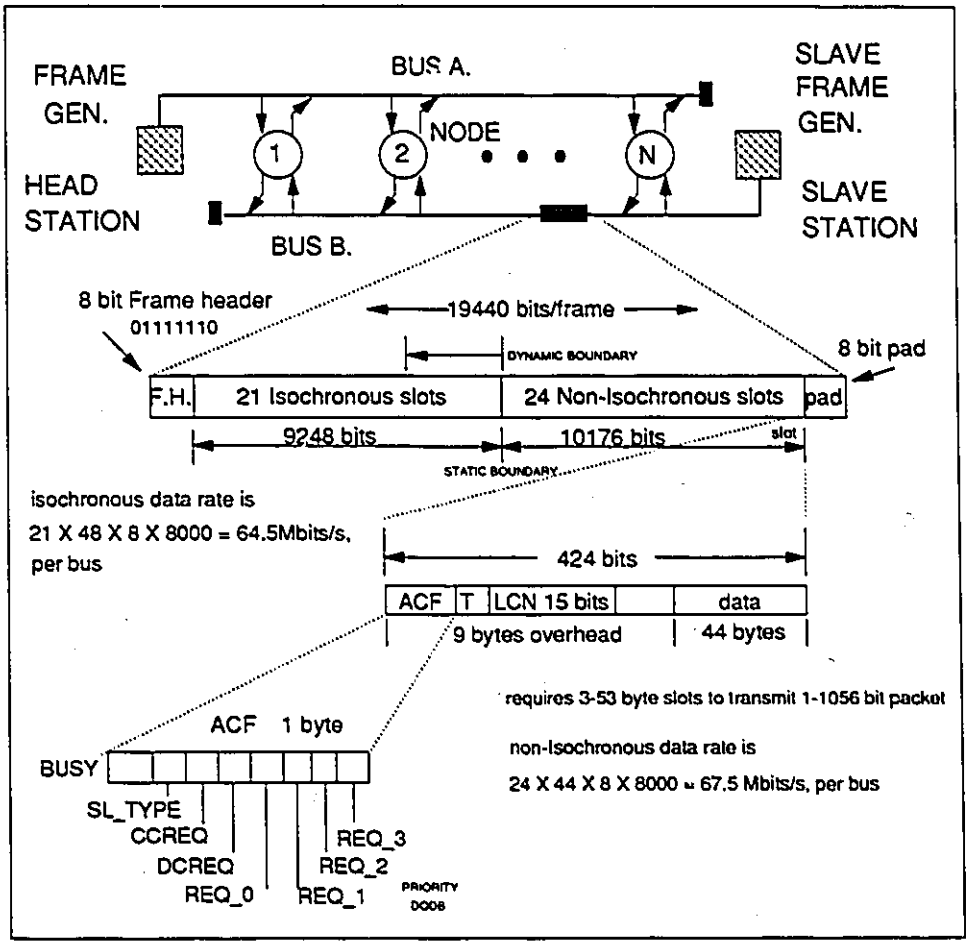


Figure 10 Simulated DQDB Slot Design. [63].

to non-isochronous slots is *Queued Arbitrated* (QA). Every slot contains a segment of 52 bytes and a one byte Access Control Field (ACF).

The Division Between Computer and Digitized Voice Communication within a Frame

In [64], a strategy is described in which $N_1 < N$ slots per frame are reserved for isochronous traffic. $N_2 = N - N_1$ slots are then allocated to non-isochronous traffic.

Unused isochronous slots in a given frame may be used, for that frame only, by non-isochronous traffic. A scheme of this type has been termed as *movable boundary* and was first proposed in [65]. An alternate strategy called a *static boundary* allocates approximately half of the total bandwidth to isochronous traffic and the other half to non-isochronous traffic. There is a trade-off between isochronous traffic being blocked (this results from greater allocation of bandwidth to non-isochronous traffic) and non-isochronous having to queue (this results from greater allocation of bandwidth to isochronous traffic).

Computer Communication

Access to QA slots is controlled by the Distributed Queueing MAC Protocol and would be used typically to provide non-isochronous services. The static boundary simulation models the Distributed Queueing protocol with 24 non-isochronous slots and their QA access mechanisms. The model takes into account the use of 21 slots for isochronous data. Each non-isochronous slot is used to transfer one segment.

It should be noted that although each non-isochronous slot contains 53 bytes of information, only 44 bytes are available for data payload use. Nine bytes from each slot are considered overhead.

Digitized Voice and Video Communication

PA slots are typically used to provide isochronous services (i.e. voice calls). Centralized access to PA slots is controlled by the Frame Generator and Slave Frame Generator. [28] proposes a distributed isochronous channel management protocol in which each station

assigns, maintains and deallocates its isochronous channels without communicating with other stations.

As shown in Figure 10, the DQDB static boundary simulation allocates approximately half the bandwidth (21 PA slots) for isochronous services. The PA segment payload is 48 bytes long which consists of 48 isochronous service bytes (21 slots supply bandwidth for approximately 1000 isochronous channels). In [64], various models for integrating voice and data are described. With PA traffic, different users would be assigned a different number of isochronous service bytes, depending on their bandwidth requirements. In Figure 10 it is assumed that each user requires the same bandwidth, one byte. Access to a PA slot may be shared among a number of isochronous service users. Each user keeps its assigned channel (same byte location within a slot and same slot location within a frame) for the duration of the call, and is blocked if no assignment is available.

The dynamic boundary simulation models the Distributed Queueing protocol for non-isochronous data, with 24–31 slots per frame. The movable boundary allows the use of PA slots by QA segments, if available. Figure 11 shows the probability distribution of the number of slots available for use by QA segments. The mean of this distribution is 24.98.

Controlled Access to a Slot within a Frame

Of the nine non-isochronous overhead bytes, only two bits are used in the simulation model. These bits are located in the ACF (Figure 10). They are the BUSY bit and the REQ_0 bit. The BUSY bit is used to indicate whether a slot is available for use (0 or reset indicates that the slot is free for use, 1 or set indicates that the slot is busy and unavailable for use). The REQ_0 bit is used to indicate whether a node wishes to queue (0 or reset

indicates no request, 1 or set indicates a request for a empty slot). The SL_TYPE bit in the ACF indicates whether the slot is a Queued Arbitrated slot (SL_TYPE = 0) or a Pre-Arbitrated slot (SL_TYPE = 1). In the simulation the SL_TYPE is considered to be always zero. Also the three remaining fields (REQ1, REQ2, REQ3) in the ACF are used for priority queuing. Only a single distributed queue is being modeled utilizing only the REQ_0 bit (note: the Priority Distributed Queue has important uses — i.e. for network signaling).

Simulated Dual Bus Addressing

By studying only network load and the Distributed Queuing mechanism, the actual simulation of point-to-point communication between any two nodes is unnecessary. As a result, addressing in the simulation is handled very simply. Dual bus addressing is simulated by nodes in the following way: nodes wishing to transmit to a node with a higher address access the forward bus, while nodes wishing to transmit to a node with a lower address transmit on the reverse bus. The simulation assigns an equal probability of sending a packet (one 1056 bit packet requires 3 slots. See Figure 10) on either the forward or reverse bus. The only deviation from this rule is at the end nodes. The first node (node 1) accesses the forward bus 100 percent of the time and the Nth node accesses the reverse bus 100 percent of the time. It should also be noted that once a slot is marked as busy it remains busy for the duration of the time it is on the bus.

Information packets greater than the slot payload size (in this case 44 bytes) must be segmented. In Figure 10, a 1056 bit packet requires a total of three physical slots to be transmitted. In the simulation, 1056 bit packets are generated at random using

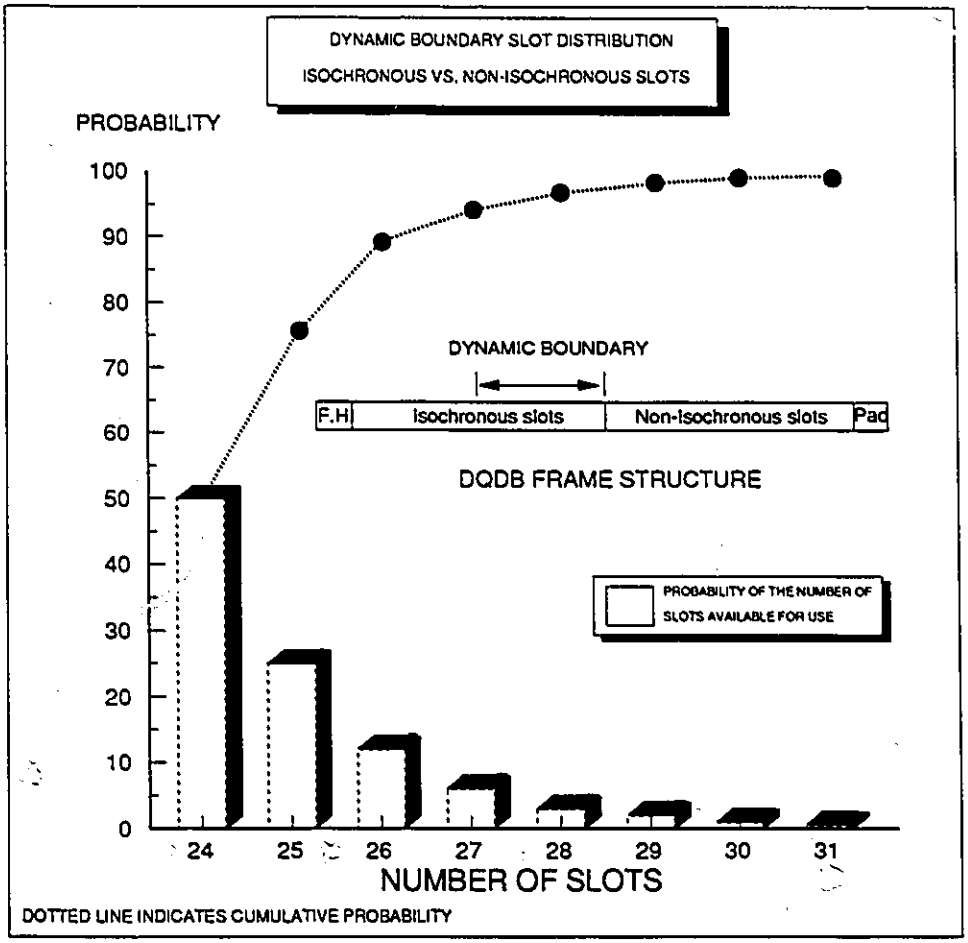


Figure 11 Dynamic Boundary Slot Probability Distribution.

the Poisson Distribution. This creates a demand for three slots on either the forward or reverse buses. Access to empty slots needed to transmit these segments is arbitrated by the Distributed Queueing Protocol which is discussed below.

Bothway Transmission of Data

A DQDB Broadcast (Bothway Transmission) or Multicast implementation allows nodes to transmit a segment on both buses. This allows all or a subset of stations to receive

segments at the same time. Advantages of such a scheme are: 1) nodes avoid the creation and maintenance of bus selection tables, and 2) nodes avoid costly reconfiguration of these tables which can result from bus/node faults or node relocation.

Fair Sharing of the Bandwidth Resource

Bandwidth Balancing (BWB) techniques are used to ensure fair sharing of the bandwidth resource between stations operating at a single priority. In [64], BWB is defined as a procedure to facilitate effective sharing of the bandwidth, whereby a node occasionally bypasses the use of empty QA slots. In [66], it is stated that BWB may be desirable in a DQDB subnetwork when the distance between any two stations exceeds 546 meters, transmission rate is 155.52 Mbps, and the offered load of all stations exceeds the bandwidth available.

Bandwidth Balancing is implemented by artificially setting a node's countdown (CD) counter to a value greater than zero following the transmission of a segment. A full description of the IEEE 802.6 standard BWB mechanism is provided in [66]. The simulation models simpler BWB mechanisms resulting in a straightforward model requiring less coding. What follows is a description of the BWB mechanisms used in the simulation model:

I. IEEE 802.6 *Default Bandwidth Balancing mechanism* (i.e. no BWB mechanism used, reset-value = 0).

II. An *Even BWB* mechanism such that the CD counter of all nodes is artificially set by the same amount after a segment transmission. The reset value of a

node's CD counter is based on half the network size.

$$Reset\ Value = N \div 2$$

$$Where:$$

$$N = Number\ of\ nodes\ in\ network$$

The goal of this mechanism is to allow all stations to receive an equal share of the bandwidth. With this scheme, bandwidth will be wasted.

III. An *Uneven BWB* mechanism such that a node's CD counter is artificially set after a segment is transmitted. The word *uneven* is used to indicate that each node has a unique reset value based on its location in the network. The reset value of a node's CD counter is based on the number of nodes between itself and the end of the bus.

$$Reset\ Value = N - L$$

$$Where:$$

$$N = Number\ of\ nodes\ in\ network$$

$$L = Relative\ position\ of\ node\ in\ network$$

This mechanism tries to distribute bandwidth evenly without waste.

4.2 DQDB PROTOCOL LOGIC DEVELOPED FOR THE SIMULATION: A DETAILED DESCRIPTION

The simulated DQDB performs the Medium Access Control (MAC) procedure for write access of QA segments into empty QA slots. The DQDB operates request counters, countdown counters and local request queue counters for access to each bus.

Distributed Queueing Variables

The simulation programs maintain two independent request counters: RREQ COUNTER for the forward bus and FREQ COUNTER for the reverse bus. Each node has a pair of these counters which are used when queueing a new Queue Arbitrated (QA) segment. If a node does not have any QA segments queued for access, the counters indicate the number of currently outstanding requests from downstream for access to the respective bus. If the node does have a QA segment queued, the counter indicates the number of requests from downstream for access to the bus, that arrived at the node after the QA segment was queued. All counters have a minimum value of zero and a maximum value of $2^{16}-1$. All counters are set to zero at start-up.

The simulation maintains two independent countdown counters for each node. The FREQ CD and RREQ CD counters are used by the Distributed Queue State Machine (DQSM), which performs the access function for QA segments. The FREQ COUNTER is associated with the FREQ CD counter and RREQ COUNTER is associated with the RREQ CD counter. Each countdown counter operates when a node has a QA segment queued for its respective bus. The counter indicates the number of outstanding requests made for access to bus x before the QA segment can be transmitted.

The simulation also maintains two independent segment counters. The FSEGMENT and RSEGMENT counters are used by the DQSM. The FSEGMENT counter keeps track of the number of outstanding segments that will require empty forward bus slots. The RSEGMENT counter keeps track of the number of outstanding segments that will require empty reverse bus slots.

Simulated Slot Access Control Field

Also shown in Figure 10, are the BUSY and REQ bits in the ACF fields in slots on the Forward and Reverse buses. The operation of the request and countdown counters (N1 RREQ COUNTER — N1 FREQ COUNTER and N1 FREQ CD — N1 RREQ CD respectively for node 1) for each bus involves read operations on the BUSY bit in the ACF for all slots on the forward bus and read operations on the REQ bit in the ACF field of all slots on the opposite bus. For each bus there is an instance of the DQSM (DQSM logic is duplicated for each bus) which controls the request and countdown counters for that bus.

A request for access to the forward bus is signaled to other nodes by a write operation performed on the REQ bit in the ACF of each slot passing on the opposite bus. The write operation stops when a REQ bit is set on the opposite bus. In the simulation only one segment is queued for a slot at a time. A slot is used when it is available for this queued segment. If the slot is unavailable for use, only then does this segment queue.

The Frame Generators

The purpose of the Frame Generator (FG) and the Slave Frame Generator (SFG) is to generate 24 empty 53 byte non-isochronous slots, every 125 microseconds (isochronous slots are not simulated). Empty Queued Arbitrated (QA) slots are generated by the FG and SFG by resetting (0) the BUSY, REQ and SL_TYPE bits. As in the case of the FG, it sends a free slot to node 1 so that node 1 can process the information in the ACF during the setup time (DQSM Figure 12). When node 1 has reached its Forward Bus Sync Time it resends the slot to node 2. At the same time node 2 is resending its slot to node 3, and so on. The SFG sends an empty QA slot to the Nth node on the opposite

bus, so that it can process the information in the ACF during the setup time. When the Reverse Bus Sync Time (the reverse bus DQSM logic not shown) is reached, the Nth node resends the slot to the (N-1)th node, and so on. The three internal node times (setup time, internal node change time, and Forward/Reverse bus sync time) are required to ensure that all counters and semaphore states are correct at specific points in time. This ensures proper synchronization of the Forward and Reverse Distributed Queue for each node. Each node in the network performs its own specific Forward and Reverse Distributed Queue functions during these time intervals.

The Distributed Queue State Machine: The Idle State

The two Distributed Queue State Machines (one for the forward bus, FDQSM and one for the reverse bus, RDQSM) for each node can be in one of three states: Idle, Countdown or Waiting. The Distributed Queueing State Machine (DQSM) is in the Idle State when it has no QA segments to be transferred on a bus. The DQSM is in the Countdown State when it has a QA segment queued for transfer on one of the two buses. The DQSM is in the Waiting State when it has a QA segment to transfer but the slot is busy and the Reverse bus REQ bit has already been set by a down stream node. In effect the DQSM waits. In all states the FDQSM observes the Reverse bus for requests generated by nodes downstream on the Forward bus. A request for a slot in the simulation is indicated by the REQ_0 bit in the ACF being set to one. The FDQSM also observes the Forward bus for empty QA slots, which are indicated by the BUSY bit being reset to zero.

The logic that depicts the Idle state for node 1 (FDQSM) is shown in Figure 12. While the FDQSM is in this state, the Distributed Queue maintains the value of the

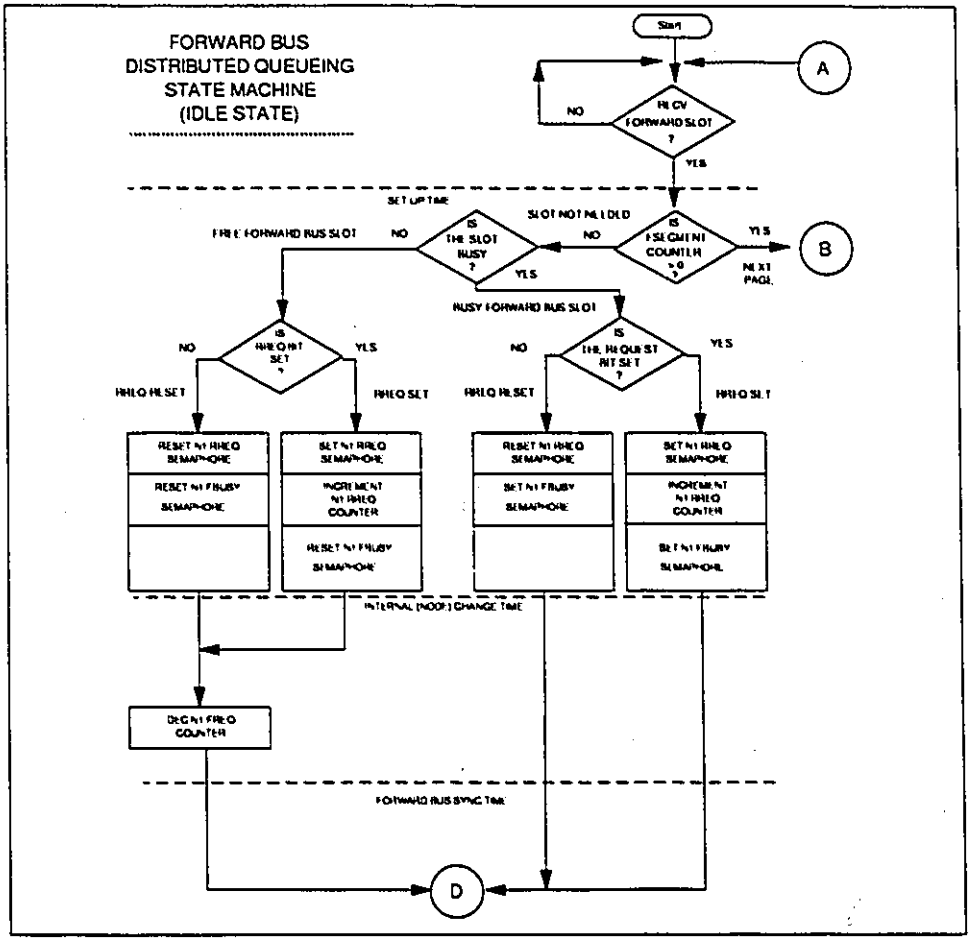


Figure 12 Forward Bus Distributed Queueing State Machine for node 1. Idle State. [63].

associated request counters by:

1. Incrementing the RREQ COUNTER for any REQ bit set (1) on the forward bus.
2. Decrementing the FREQ COUNTER for each slot which passes on the forward bus with the BUSY bit reset (0).

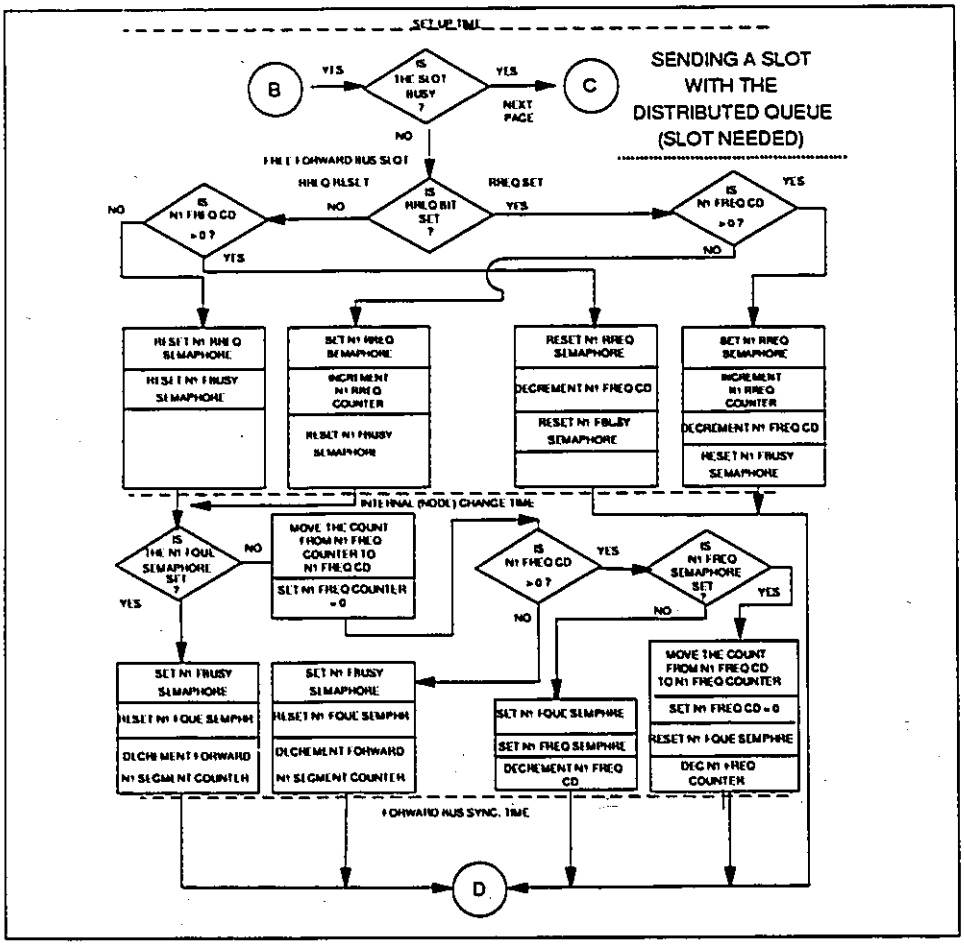


Figure 13 FDQSM for node 1. Countdown State. [63].

The FDQSM remains in the IDLE state until it is requested to queue a QA segment for transfer (FSEGMENT > 0).

The Distributed Queue State Machine: The Countdown State

When a QA segment is queued for access to the Forward bus, the associated instance of the FDQSM is in the countdown state (Figure 13). The FDQSM remains in the countdown state until all segments are written to empty QA slots and the FSEGMENT

count is equal to zero. While in the countdown state all REQ CD counters are maintained by the following actions.

If an empty QA slot is received on the Forward bus and the FREQ counter equals 0 then the FDQSM marks the QA slot as busy by setting the BUSY bit and writing the QA segment to the QA slot (Figure 13). It is important to note that the FDQSM still maintains the value of the associated request counters in this state, as discussed above.

If an empty QA slot is received and the FREQ CD is greater than zero, the FREQ CD counter is decremented by 1. The slot is then resent to the next node, empty.

If a busy QA slot is received and the node has a segment to send, (FSEGMENT > 0) it sets the REQ_0 bit on the opposite bus (Figure 14). Each DQSM on the Forward and Reverse bus is uniquely associated with the REQ on its "opposite" bus.

The Distributed Queue State Machine: The Waiting State

Figure 14 depicts a waiting to queue state. In this state a busy QA slot is received on the Forward bus and the REQ bit is set on the Reverse bus. The FDQSMs can neither send a QA segment nor set the REQ bit on the opposite bus. In this case the FDQSMs resend the forward slot and the RDQSMs resend the reverse slot to the next node without taking any action. In other words, the node is actually *waiting to queue.*

4.4 BASIC ASSUMPTIONS USED IN THE DQDB SIMULATION

Computer network designs follow a hierarchical approach (OSI reference model, [57] and [56]). The OSI model is a reference model for this simulation work.

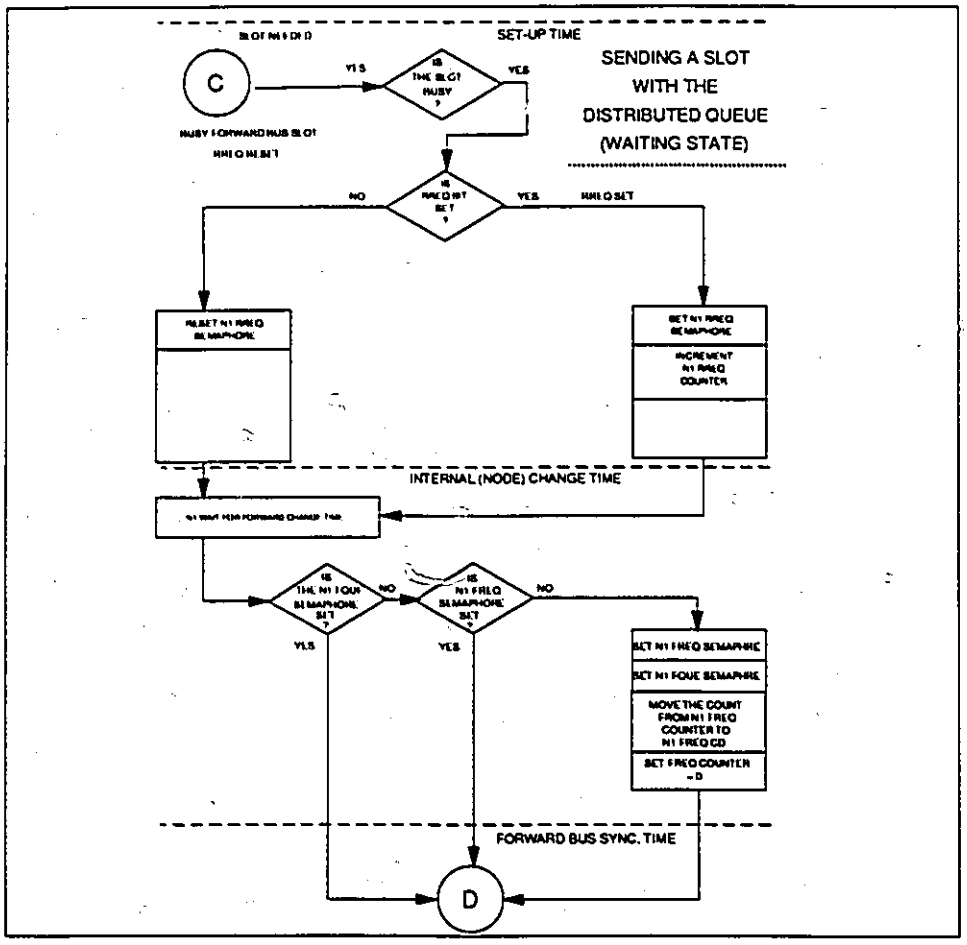


Figure 14 FDQSM for node 1. Waiting State. [63].

In the simulation model there are four layers: Physical, Data link (including Medium Access Sub-Layer), Transport and Application Layer (User). For every send-receive pair of nodes A and B on the forward or reverse bus, the model can be viewed as in Figure 16. Important assumptions used in the simulation model are described below.

Physical Layer Assumptions

All nodes are assumed to be homogeneous. The physical links which interconnect nodes

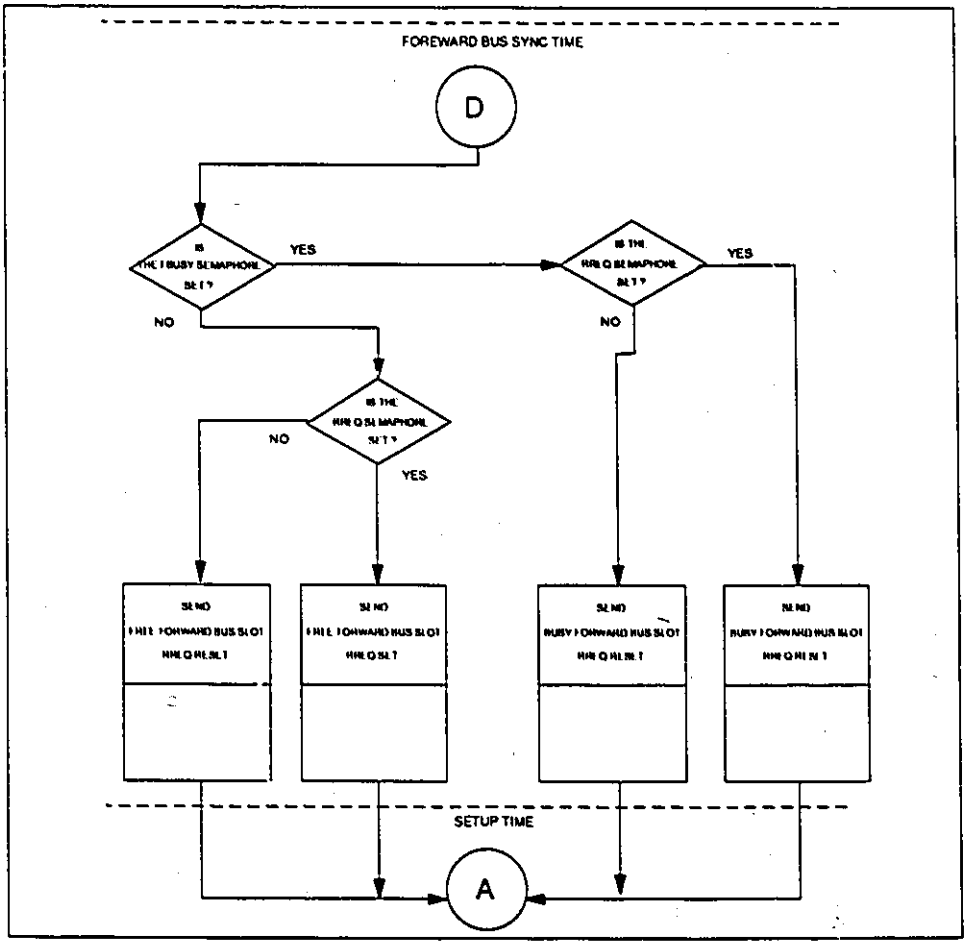


Figure 15 FDQSM for node 1. Slot Retransmission Logic. [63].

are assumed to employ Wavelength Division Multiplexing (WDM). The bit rate of both channels is 155.520 Mbps (CCITT G.707-9 SDH). Stations accessing a given bus are assumed to span a distance equivalent to the transmission time of one 53 byte slot (approx. 546 meters assuming no station latency). Finally, frames of size 19940 bits are generated every 125 microseconds by the frame and slave frame generators.

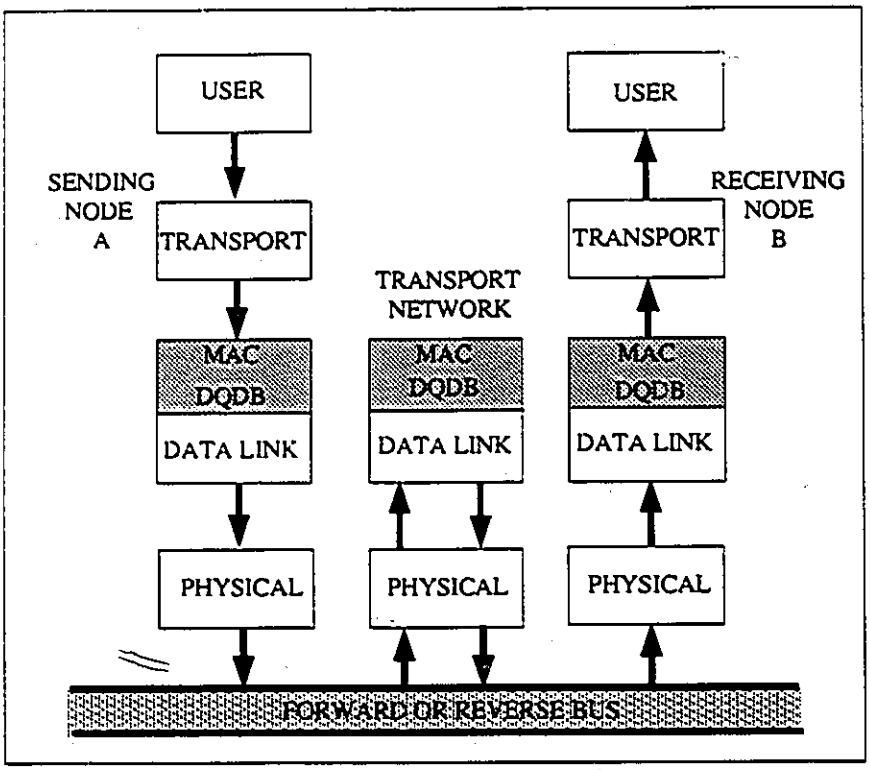


Figure 16 DQDB Layered Simulation Model.

Data Link Layer Assumptions

The purpose of this layer is to detect transmission errors and re-transmit unacknowledged segments. In this simulation the error rate is assumed to be 4.0×10^{-11}/bit therefore, an error free channel is assumed.

Medium Access Sub-Layer Assumptions

For a multiaccess channel, the key issue is to determine who gets use of the channel when there is competition for it. The simulation describes the queued arbitration access method and takes into account the load of approximately 1000 voice channels (21 slots).

In the static boundary simulation model it is assumed that 1000 voice channels are simultaneously being used for the duration of execution of the simulation. Hence, the model utilizes a static boundary scheme which allocates approximately half of the total network bandwidth to non-isochronous traffic (24 QA slots). In the movable boundary simulation model unused PA slots are allocated for use by QA segments. Figure 11 shows the probability distribution of the number of slots available for use by QA segments.

The simulation model assigns all segments the same priority.

Transport Layer Assumptions

The simulation assumes that the Transport Layer provides the User Layer with an error free point-to-point connection. As such, every node is assumed to have a logical connection to every other node in the network. Logical connections are assumed to exist throughout the execution of the simulation.

User (Application) Layer Assumptions

The User Layer of every node generates fixed length packets (1056 bits) addressed to any node in the network with equal probability. Owing to the dual bus nature of this network, packets are transmitted on the forward and reverse bus with equal probability (exception being the end nodes) in the Addressing simulation model and on both buses with a probability = 1 in the Broadcasting simulation model.

The mean time between packet generation by each node is assumed to be exponentially distributed with a mean ranging from 4.0×10^{-3} second to 1.67×10^{-5} second (2,500 PKTs/second to 60,000 PKTs/second).

General Assumptions

The simulation models the DQDB MAC protocol only and not the actual network. It is assumed that there is no limit to the buffering capacity of the queues. Also the simulation uses the first 5.12 microseconds (one slot period) to warm up.

4.5 RESULTS AND CONCLUSIONS OF THE DQDB TESTBED INVESTIGATION

Results of the Static Boundary and Default Bandwidth Balancing Techniques

Figure 17 depicts the results of increasing load on a 4, 8 and 16 node network. Figure 17A depicts the average slot queueing time, increasing exponentially with an increase in load. Figure 17B depicts the similar behavior for average packet queueing time. In the simulation packet queueing time is the aggregate of three slot queueing times. The slight increase in time is due the fact that the arrival of the first slot is random (REQ counters can grow to a large amount), with the use of the remaining two slots dependent on the first (the time interval between slots only allows a limited growth in the REQ count). This shows that transmitting very short (control) messages can take almost as long as packet sized messages. Figures 17A and 17B also show that under light load access delay is small. This is in contrast to the token ring where greater delay results from token transfers over large areas [29].

Figure 17C depicts slot utilization approaching 100 percent with an increase in load. It also shows that the utilization does not decrease when the networks are overloaded

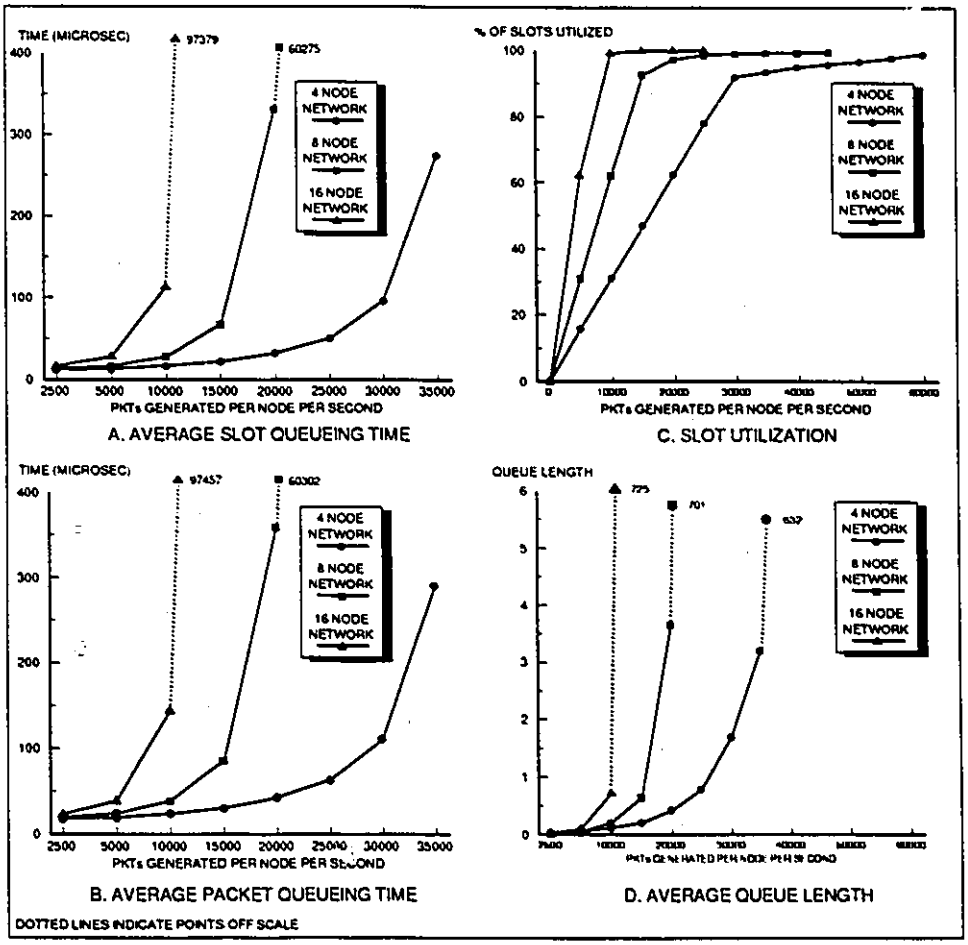


Figure 17 Results of the Static Boundary and Default Bandwidth Balancing Techniques. [30].

(CSMA/CD shows a decrease in throughput with very long cables, short messages and when the network load exceeds its maximum operating range [57]).

Figure 17D depicts queue length increasing exponentially with an increase in load. When Figure 17D is compared to Figure 17C, queue length increases without bound when slot utilization approaches 100 percent.

Figures 18A and 18B depict the average packet queueing time by a node's location

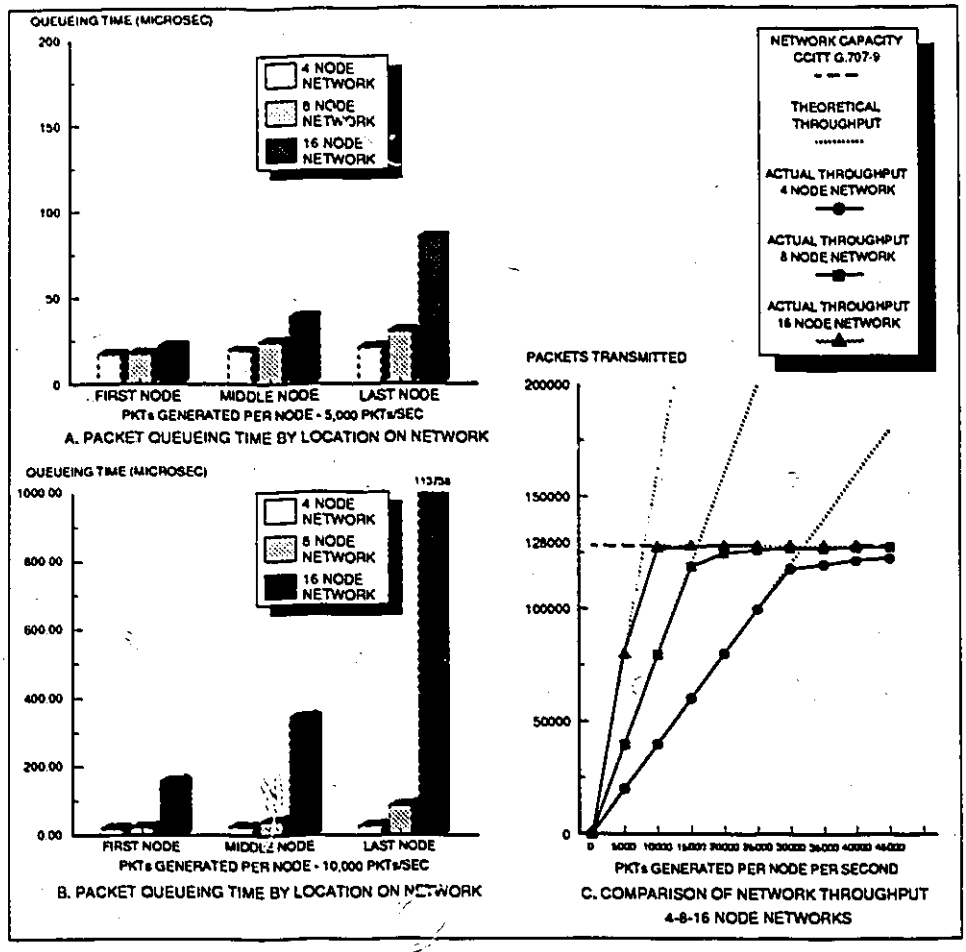


Figure 18 Results of the Static Boundary and Default Bandwidth Balancing Techniques. [30].

in a network under a load of 5,000 and 10,000 packets/sec respectively. These graphs show that network size and a node's position on the bus influence access delay. The larger the network and the further a node is away from the frame generator, the greater is the access delay. Figure 18B demonstrates that this trend increases in severity as network load increases.

Figure 18C depicts the CCITT G.707–9 network capacity and compares the theoret-

ical and actual throughput of 4, 8 and 16 node networks. Figure 18C demonstrates that the Distributed Queueing protocol effectively uses all capacity. It closely approximates the theoretical capacity until the CCITT G.707–G capacity is reached. The maximum operating range of a 4, 8 and 16 node network is demonstrated to be approximately 40,000, 20,000 and 10,000 packets per second respectively. This diagram can also be used to predict the throughput of these networks as the bit rate increases and to predict the maximum operating range of larger networks.

Comparison of the Address and Broadcast Techniques

Figure 19 compares the results of the Broadcast technique (segment transmission on both buses) to the standard DQDB Addressing technique (segment transmission on one bus). Figure 19A depicts the influence of node location on packet queueing time. It can be noted that the bandwidth in a 16 node network using the Addressing technique, is shared fairly. With the Broadcast technique, the further a node is away from a frame generating station, the higher the delay.

Figure 19B depicts the slot utilization of a 4, 8, and 16 node network using the Addressing technique and a 16 node network using the Broadcast technique. Figure 19B shows that the Broadcast technique utilizes 100 percent of the bandwidth with increasing load. It shows that the Broadcast network reaches 100 percent channel utilization (5,000 PKTs/second) much faster than the 4, 8, and 16 node Address networks (40,000 PKTs/second, 20,000 PKTs/second and 10,000 PKTs/second respectively). Figure 19B also shows that the Broadcast network continues to utilize 100 percent of the bandwidth when the network is overloaded.

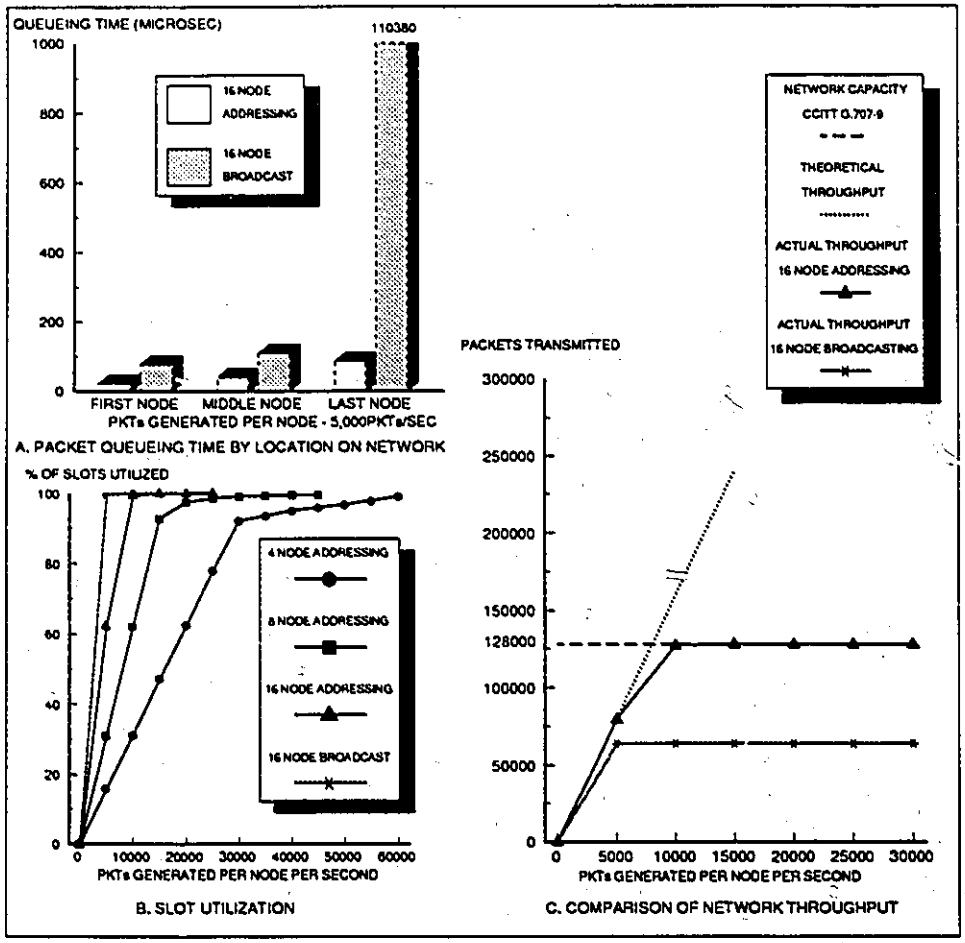


Figure 19 Comparison of the Address and Broadcast Techniques.

Figure 19C depicts the CCITT G.707–9 network capacity and compares the theoretical and actual throughput of a 16 node Address network to a 16 node Broadcast network. Figure 19C shows that throughput of the 16 node Broadcast network is half that of the 16 node Address network [30]. The graph clearly shows that the implementation of the simpler DQDB network using Broadcasting is less efficient than an Addressed DQDB network.

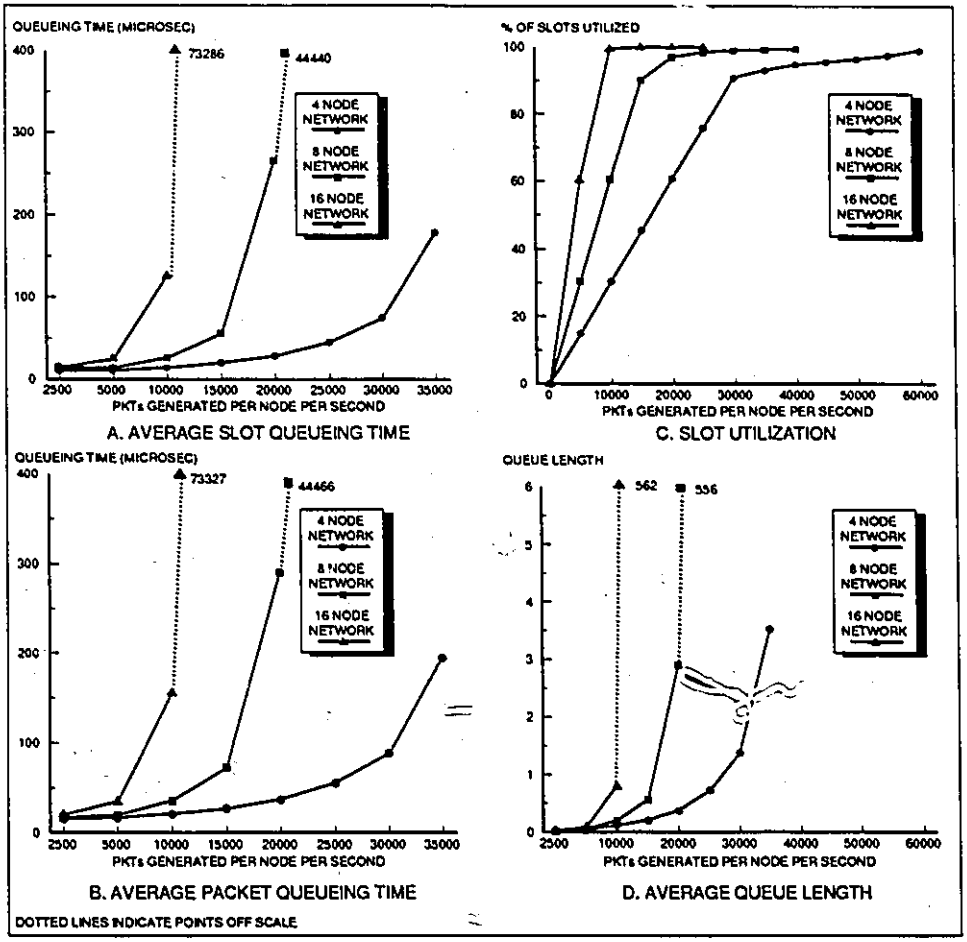


Figure 20 Results of Combining the Dynamic Boundary and Default Bandwidth Balancing Techniques.

When Figure 19A is compared to Figure 19B and Figure 19C (the 16 node Broadcast network reaches the overload point at 5,000 PKTs/sec) it demonstrates that effective sharing of the bandwidth becomes a problem when a DQDB network becomes overloaded. This implies that a bandwidth balancing mechanism should be used when channel utilization approaches 100 percent.

Results of Combining the Dynamic Boundary and Default Bandwidth Balancing Techniques

Figure 20 depicts the results of increasing load on a 4, 8, and 16 node DQDB network using the Dynamic Boundary technique. As with the 4, 8, and 16 node DQDB network using the Static Boundary, Figures 20A and 20B depict the average slot queueing time and average packet queueing time, increasing exponentially with an increase in load. In the simulation, packet queueing time is the aggregate of three slot queueing times. The slight increase in total time is due to the fact that the arrival of the first slot is random, with the use of the remaining two slots dependent on the first.

Figure 20C depicts slot utilization of the Dynamic Boundary technique approaching 100 percent with an increase in load (this was also the case with the Static Boundary technique). It also shows that channel utilization does not decrease when the three networks are overloaded.

Figure 20D depicts queue length increasing exponentially with an increase in load. When Figure 20D is compared to Figure 20C, it can be seen that queue length increases without bound when slot utilization approaches 100 percent.

Figure 20 shows that the Dynamic Boundary technique reduces both the average queue length and access delay when compared to networks of the same size utilizing the Static Boundary technique (Figure 17). This result also compares well with those given in [67].

Figure 21A (Dynamic Boundary with Default Bandwidth Balancing) shows that the 4, 8, and 16 node networks utilizing the Dynamic Boundary technique also increases

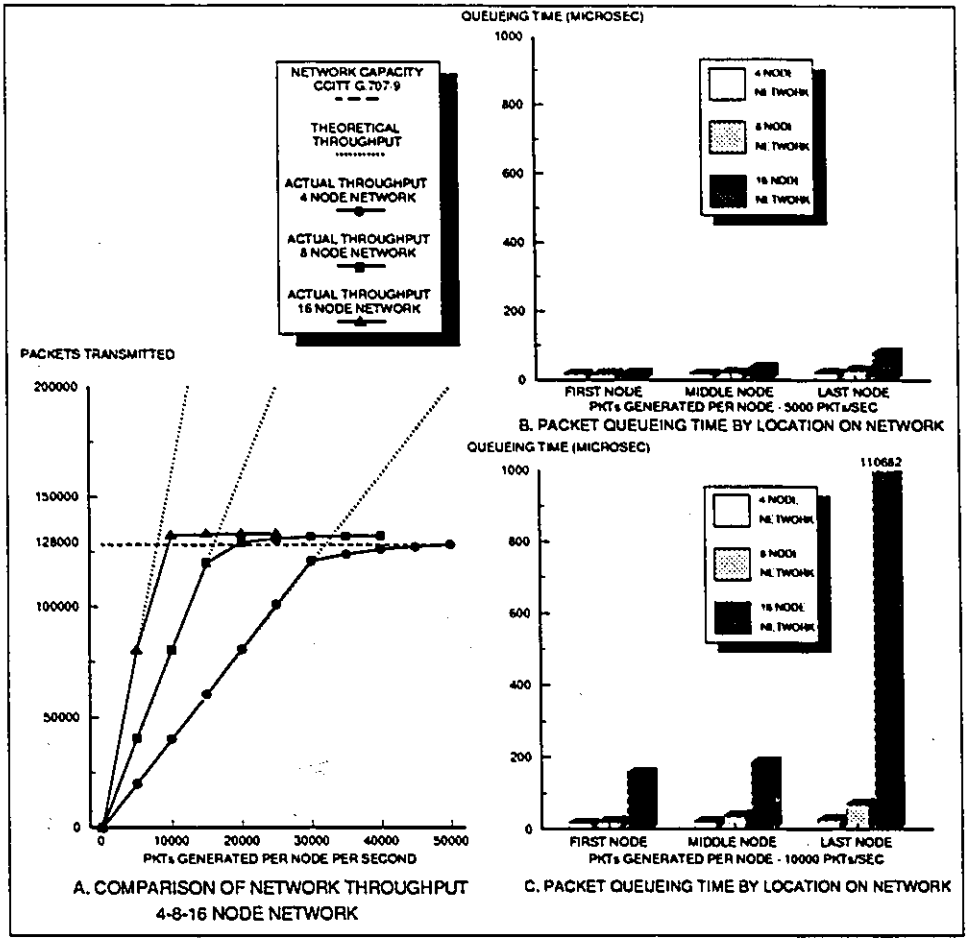


Figure 21 Results of Combining the Dynamic Boundary and Default Bandwidth Balancing Techniques.

throughput above that of the Static Boundary technique (maximum throughput = 128,000 PKTs/sec. [30]). The results show that the Dynamic Boundary makes more efficient use of the transmission link. This simulation confirms the conclusions arrived at by queueing analysis of the dynamic boundary [64].

Figure 21A also depicts the CCITT G.707–9 network capacity and compares the theoretical and actual throughput of 4, 8 and 16 node networks. Figure 21A demonstrates

that the Distributed Queueing protocol (using the Dynamic Boundary technique) effectively uses all capacity. It closely approximates the theoretical capacity and exceeds the CCITT G.707–G capacity when the network is highly loaded. The maximum operating range of a 4, 8 and 16 node network has increased to approximately 50,000, 25,000 and 13,000 packets per second respectively when compared to Figure 18. This diagram can also be used to predict the throughput of these networks as the bit rate increases and to predict the maximum operating range of larger networks.

Figures 21B and 21C depict the average packet queueing time by a node's location in a network under a load of 5,000 and 10,000 packets/sec respectively. These graphs also show (when compared to the Static Boundary results in Figure 18A and Figure 18B) that network size and a node's position on the bus influence access delay. The larger the network and the further a node is away from the frame generator, the greater is the access delay. Figure 21C demonstrates that this trend increases in severity as network load increases.

Results of Combining the Static Boundary and Even Bandwidth Balancing Techniques

Figure 22 depicts the results of increasing load on a 4, 8, and 16 node DQDB network using the Even Bandwidth Balancing technique. Figures 22A and 22B depict the average slot queueing time and average packet queueing time increasing exponentially with an increase in load. When Figures 22A and 22B are compared to Figures 17A and 17B, under light loads, they indicate that channel access is no longer immediate with the Even Bandwidth Balancing technique. Comparison of these graphs also reveals that queueing

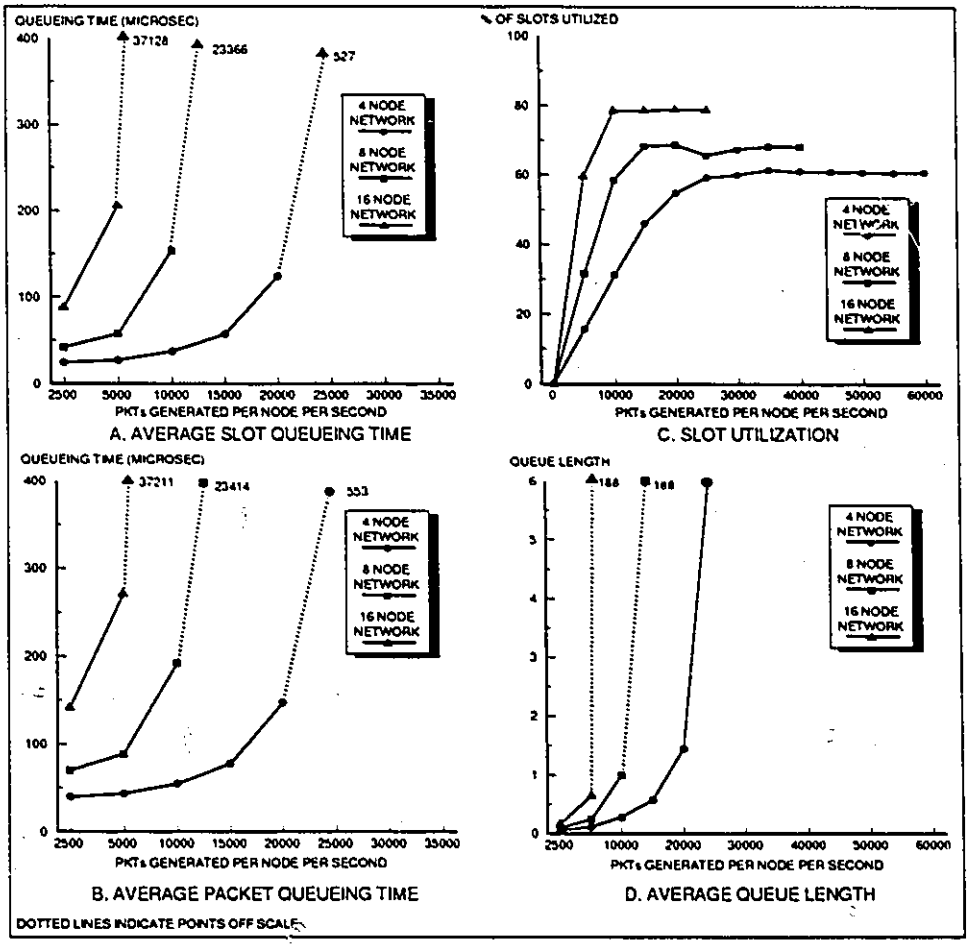


Figure 22 Results of Combining the Static Boundary and Even Bandwidth Balancing Techniques.

time is now dependent on the amount of nodes connected on the network. The comparison also shows that queueing time of the first segment of a packet is no longer random. It is now dependent on the reset value of the CD counter.

Figure 22C shows that slot utilization of the 4, 8, and 16 node networks using the Even Bandwidth Balancing technique no longer utilizes 100 percent of the available bandwidth. It shows that the Even Bandwidth Balancing technique reduces channel

efficiency. It also shows that slot utilization is dependent on network size. This implies that if the amount of nodes connected to the network is increased, slot utilization will approach 100 percent.

A comparison between Figure 22D and Figure 17D shows that the Even Bandwidth Balancing technique causes queue lengths to increase.

Figure 23A shows that with the 4, 8, and 16 node networks utilizing the Even Bandwidth Balancing technique, throughput is reduced from that of the Static Boundary utilizing the default Bandwidth Balancing technique (128,000 PKTs/second). It also shows that throughput is dependent on network size.

Figure 23B and 23C depict the average packet queueing time by a node's location in a DQDB network utilizing the Even Bandwidth Balancing technique under a load of 5,000 PKTs/second and 10,000 PKTs/second respectively. Figure 23B shows that access delay is dependent on network size. It also shows that channel access under light loads is no longer immediate. Figures 23B and 23C show that this technique forces nodes that are close to a frame generating station to wait longer than nodes which are not. This is the opposite effect from what was seen in Figure 18A and Figure 18B. The results show that the Even Bandwidth Balancing technique shares the bandwidth resource more evenly than that of the Default Bandwidth Balancing technique.

Results of Combining the Static Boundary and Uneven Bandwidth Balancing Techniques

Figure 24 depicts the results of increasing load on a 4, 8, and 16 node DQDB network using the Uneven Bandwidth Balancing technique. Figures 24A and 24B depict the

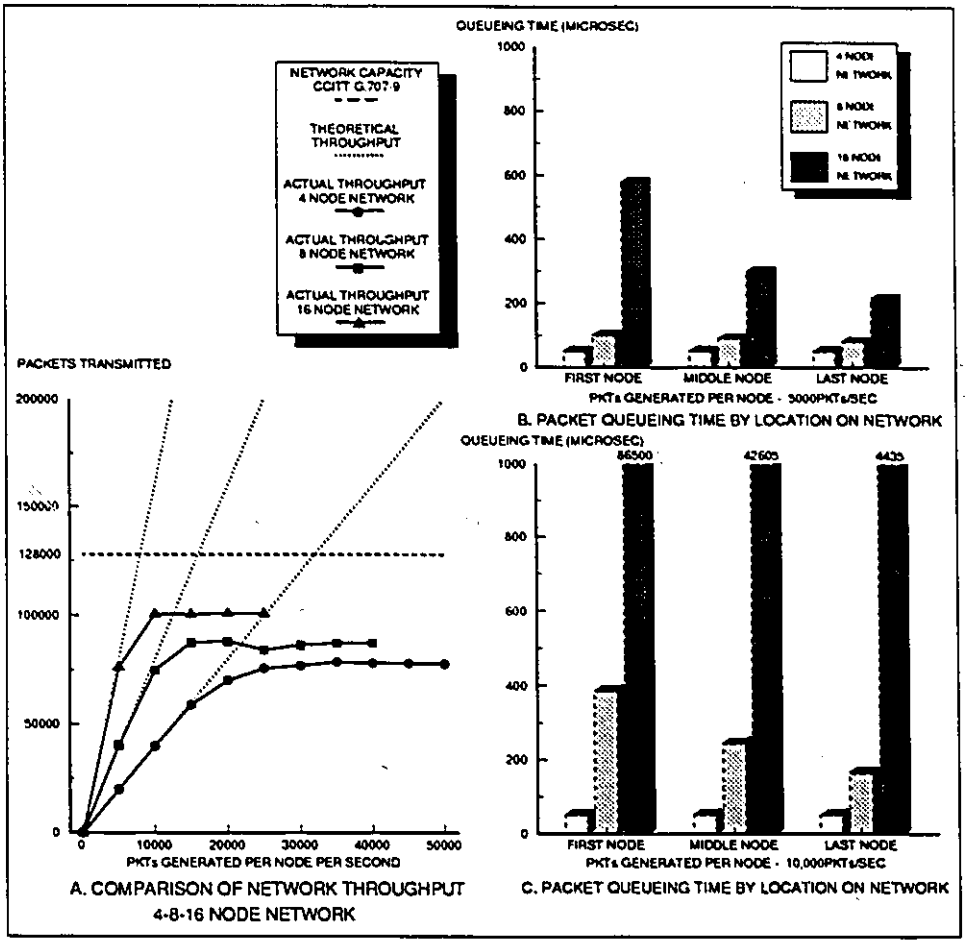


Figure 23 Results of Combining the Static Boundary and Even Bandwidth Balancing Techniques

average slot queueing time and average packet queueing time increasing exponentially with an increase in load. When Figures 24A and 24B are compared to Figures 17A and 17B, under light loads, they indicate that channel access is no longer immediate with the Uneven Bandwidth Balancing technique. Comparison of these graphs also reveals that queueing time is now dependent on the amount of nodes connected on the network.

Figure 24C shows that slot utilization of the 4, 8, and 16 node networks using the

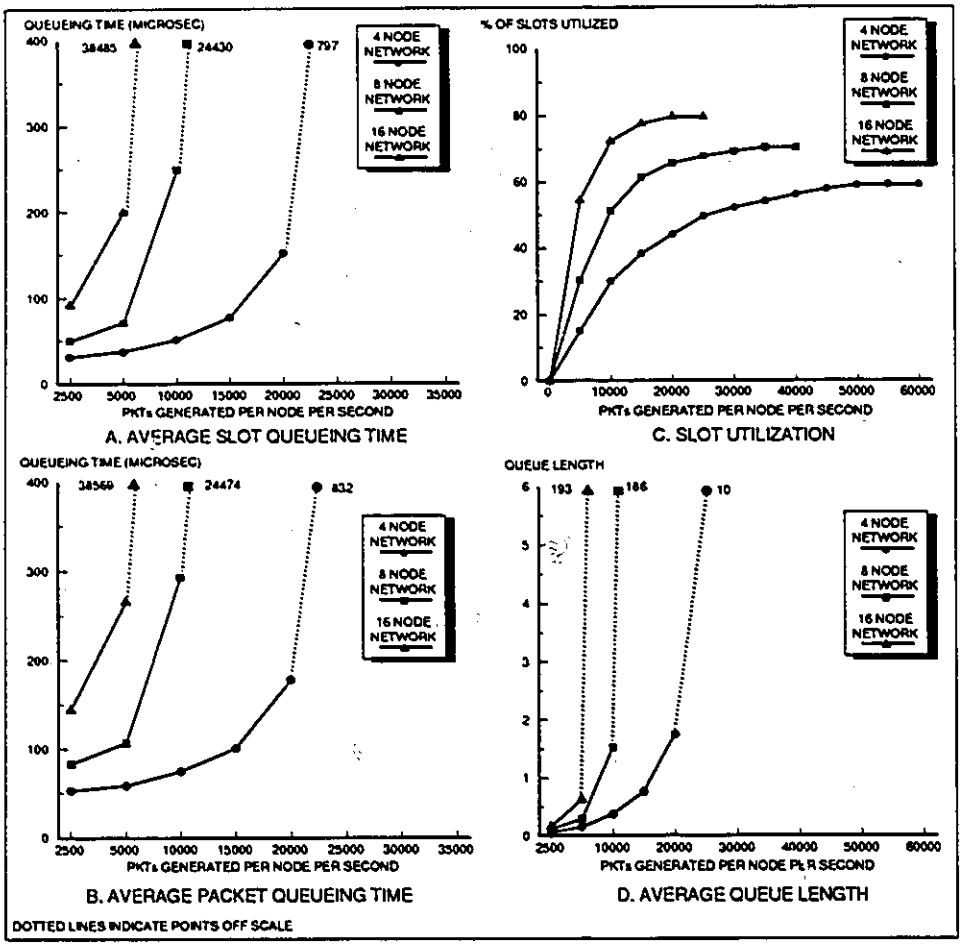


Figure 24 Results of Combining the Static Boundary and Uneven Bandwidth Balancing Techniques.

Uneven Bandwidth Balancing technique does not utilizes 100 percent of the available bandwidth (also seen in Figure 22). There is a slight decrease in Channel efficiency when compared to Figure 22C. It also shows that slot utilization is dependent on network size.

Figure 24D shows that the Uneven Bandwidth Balancing technique also causes queue lengths to increase.

Figure 25A shows that with the 4, 8, and 16 node networks utilizing the Uneven

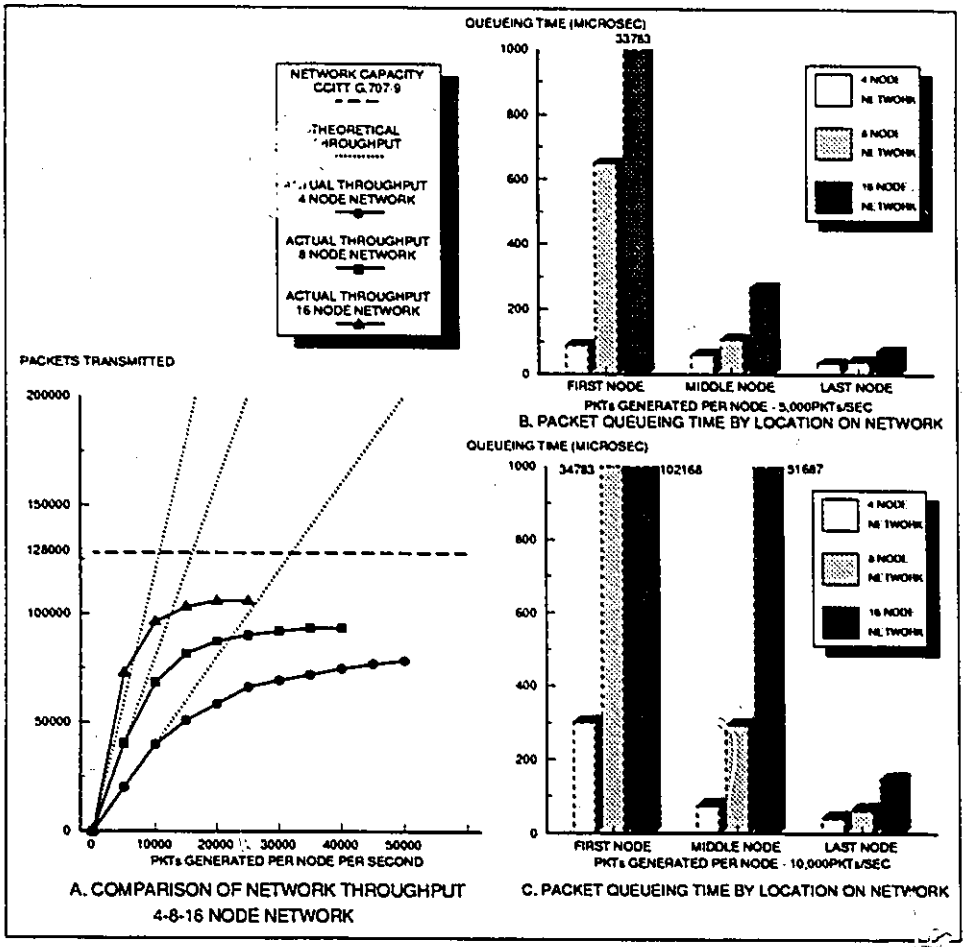


Figure 25 Results of Combining the Static Boundary and Uneven Bandwidth Balancing Techniques.

Bandwidth Balancing technique, throughput is also reduced from that of the Static Boundary utilizing the default Bandwidth Balancing technique (128,000 PKTs/second). As with the Even Bandwidth Balancing technique, throughput is dependent on network size.

Figures 25B and 25C depict the average packet queueing time by a node's location in a DQDB network utilizing the Static Boundary with an Uneven Bandwidth Balancing

technique under a load of 5,000 and 10,000 PKTs/second respectively. The goal of this technique is to force the nodes closest to a frame generating station to bypass a number of slots equal to the number of nodes downstream from it. Figure 25B shows that this technique forces nodes in larger networks to wait much longer than nodes in smaller networks. It also shows that channel access under light loads is no longer immediate. This also implies that bandwidth balancing mechanisms should be used only when channel utilization approaches 100 percent.

Figure 25C demonstrates that as the 16 node network operates in the overload region, the Uneven Bandwidth Balancing technique does not perform as well as the Even Bandwidth Balancing technique. The results show that Uneven Bandwidth Balancing is not an effective technique for sharing bandwidth in an overloaded homogeneous environment. The goal of the Uneven Bandwidth Balancing technique (share the bandwidth resource with no waste) has not been realized.

Results of Combining Dynamic Boundary and Even Bandwidth Balancing Techniques.

Figure 26 depicts the result of combining the Dynamic Boundary and Even Bandwidth Balancing techniques on a 4, 8, and 16 node network. When Figures 26A and 26B are compared to Figures 22A and 22B, access delay is shown to be reduced due to the utilization of unused isochronous slots. Figure 26C demonstrates an increase in slot utilization over Figure 22C. Figure 26D shows a decrease in queue length compared to Figure 22D. It should be noted that decreases in access delay and queue length, and increases in slot utilization are dependent on the Dynamic Boundary slot probability

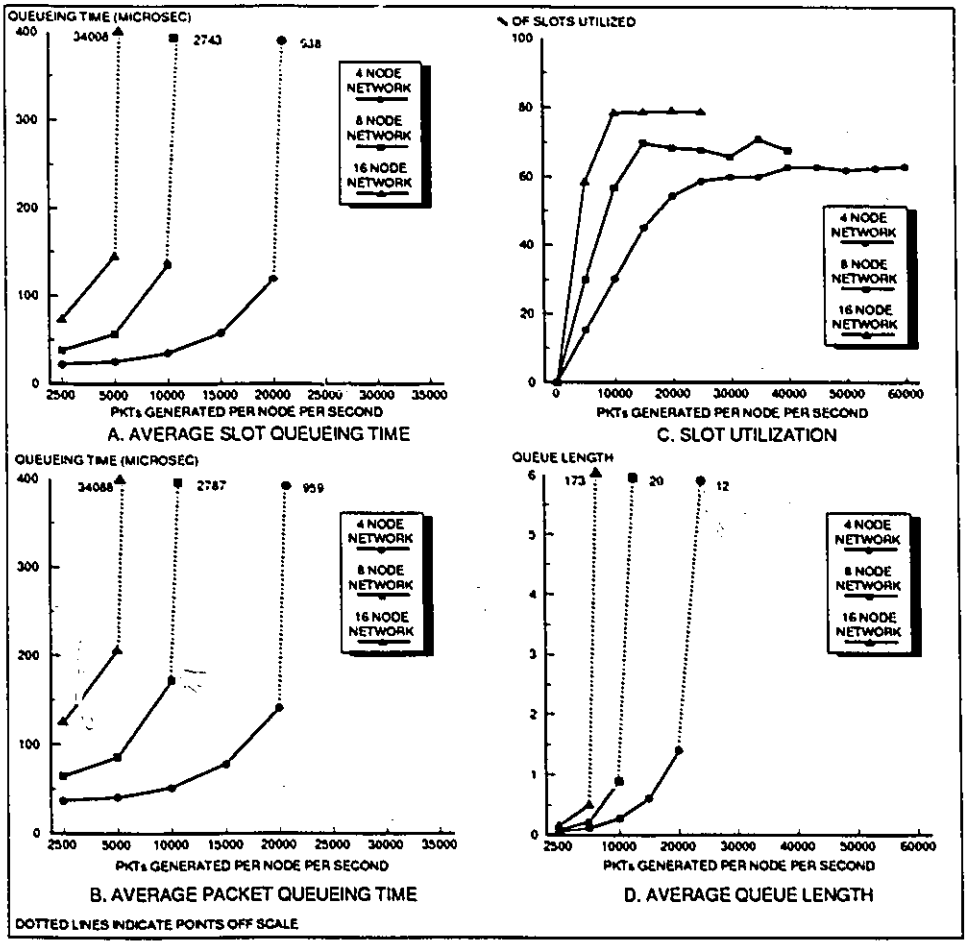


Figure 26 Results of Combining Dynamic Boundary and Even Bandwidth Balancing Techniques.

distribution (Figure 11). If the simulation is run with a higher mean value of the probability distribution, more favorable results can be expected.

Figure 27A shows that the Dynamic Boundary technique had a favorable effect on the throughput of all three networks. Figures 27B and 27C show that under homogenous network loading the Even Bandwidth Balancing technique produces the opposite effect from that of the Default Bandwidth Balancing technique. It also shows that this technique

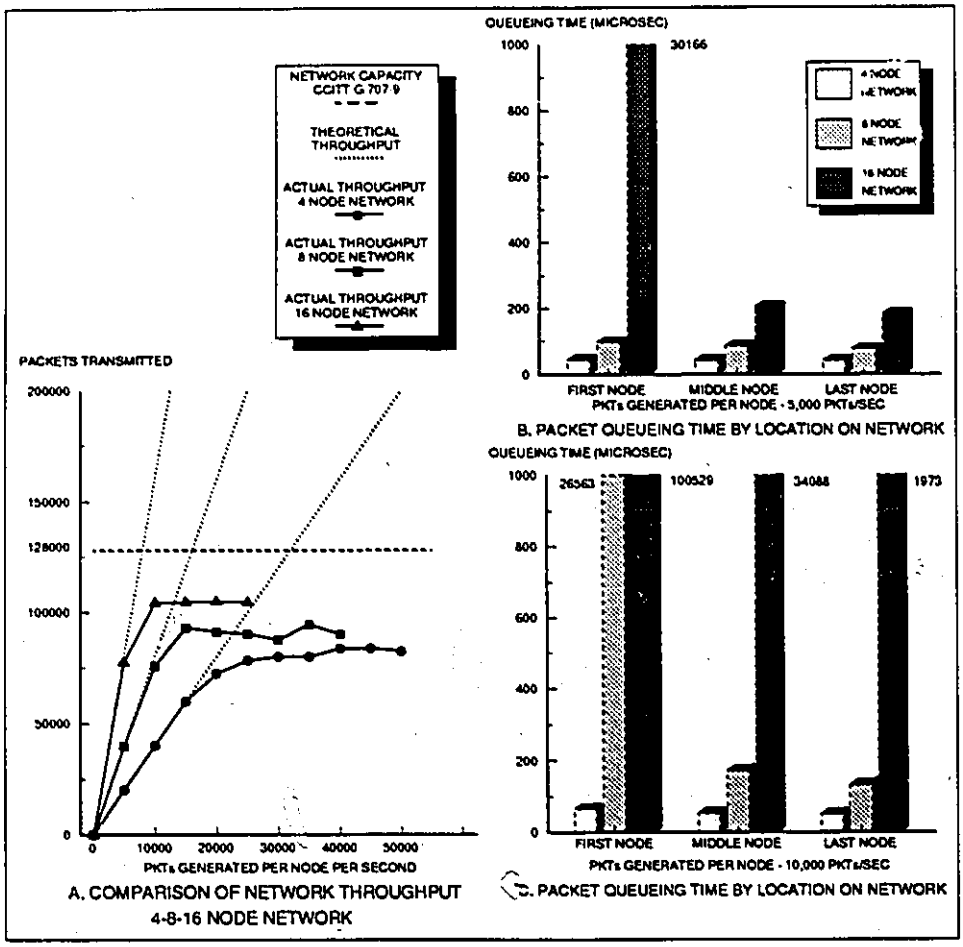


Figure 27 Results of Combining Dynamic Boundary and Even Bandwidth Balancing Techniques.

fails to evenly share the bandwidth capacity when the offered load is heterogenous.

CHAPTER 5

THE INVESTIGATION OF THE SECOND NETWORK MODEL: THE HYPERCUBE TESTBED

INTRODUCTION AND OVERVIEW

The Hypercube Topology

The sites in a computer communication system can be connected physically in a variety of ways. The various topologies are represented as graphs whose nodes correspond to sites. An edge from node A to node B corresponds to a direct physical connection between two sites. Some of the most common configurations are depicted in Figure 28. The major differences among these configurations involve [62]:

- Installation cost. The cost of physically linking the sites in the system.
- Communication cost. The cost in time and money to send a message from site A to site D.
- Reliability. The frequency with which a link or site fails.

In most cases it is just not feasible to fully-connect all sites together as depicted in Figure 28 (fully connected network). To solve this problem, other network topologies

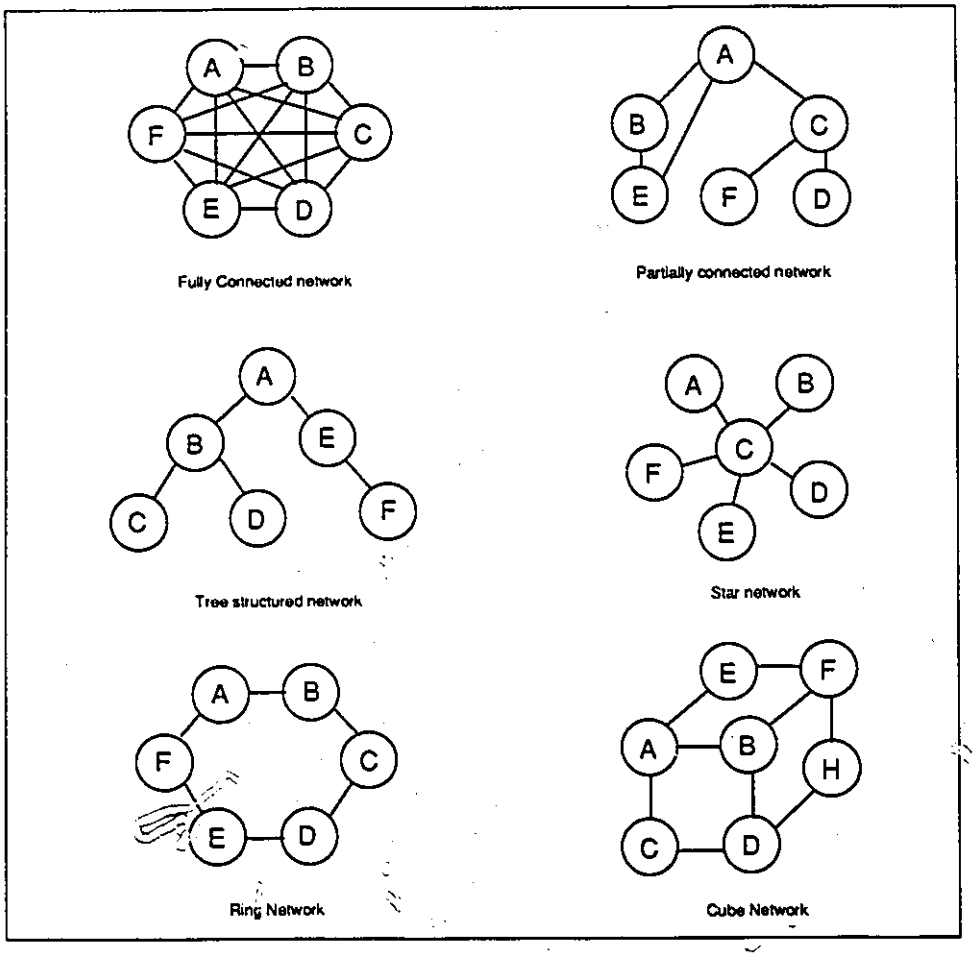


Figure 28 Network Topologies

are used (i.e. ring network) that provide for the sharing of transmission facilities among many sites. This reduces the installation cost incurred by any pair of sites at the expense of increasing communication cost.

One network topology that is a compromise between the fully connected network and the ring network is the Hypercube. The Hypercube has highly desirable properties, such as low communication diameter, high connectivity, fault tolerance, and is scalable

to thousands of nodes. A disadvantage of this type of topology is that the number of computing nodes in a Hypercube network must always be a power of two.

Definition of a Hypercube Network

The Hypercube was first proposed by [68]. A Hypercube is an n - dimensional Boolean cube with 2^n nodes. Each node has an address $[a_0, \ldots, a_{n-1}]$ where $a_i \in \{0, 1\}$.

A node x is connected to a node y if the hamming distance is equal to 1. For example, in a four-dimensional cube, node 0000 is connected to 0001, 0010, 0100 and 1000. An example of a three-dimensional cube is given in Figure 28.

The main characteristics of the Hypercube topology are low cost data transmission between sites (low communication diameter) and the topology is fault tolerant (more than one path between any pair of communicating nodes).

Two general approaches are used to transmit information on the above mentioned topologies. They are known as circuit switching and packet switching. Circuit switching is almost always used for telephone networks. Packet switching is almost universally used for computer networks. Only the packet switching technique is implemented in the model.

An Overview of the Hypercube Model

The OSI model is a reference model for this simulation work. Normally the OSI model has seven layers. However, simulation of all seven is impractical. To reduce the complexity of the simulation only four layers are modeled. The layers simulated are the: Application (User) Layer, Transport Layer, Network Layer and Physical Layer. For

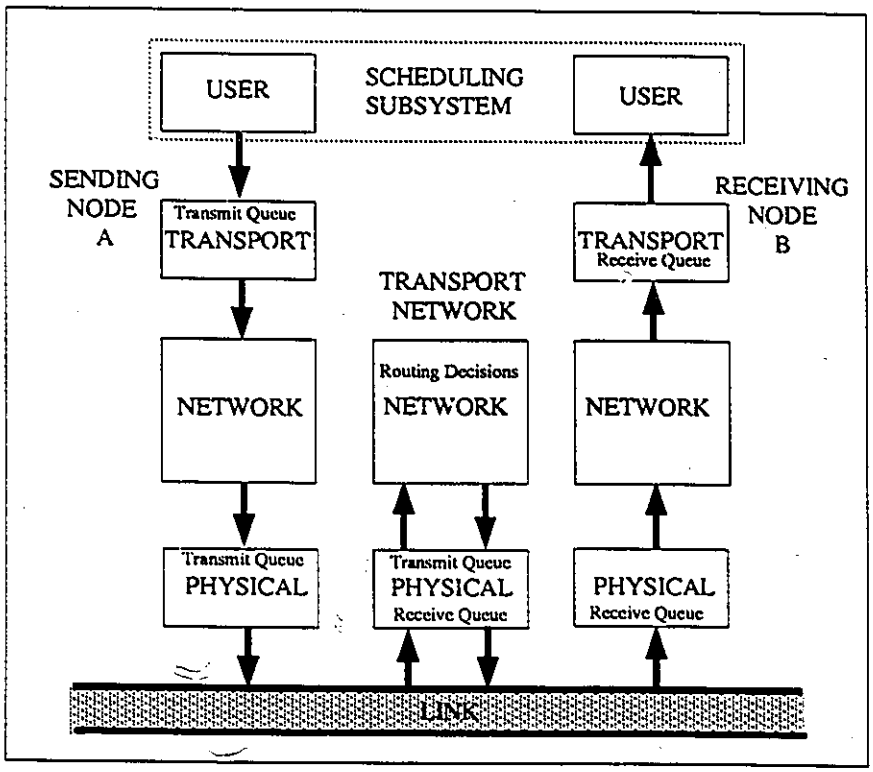


Figure 29 Hypercube Layered Simulation Model.

every send-receive pair of communicating nodes A and B, the model can be viewed as in Figure 29.

In the Hypercube model, the Scheduling Subsystem[5] of a DDMS (see Section 2.1) is implemented at the user level. The Scheduler interacts with the network communications facilities of each node involved with the query in order to transmit data between nodes as determined by the distribution strategy. The Scheduler coordinates the various schedules in the strategy so that the query response is presented at the result node.

[5]At this point, the Query Processing Subsystem is not modeled. However it will be added at a later date.

Optimization algorithms (Algorithms PARALLEL, SERIAL, GENERAL and JOIN) are executed outside the model as a separate exercise

The user level continually generates packets. Each of these packets needs to be transmitted to some other node in the network. When packets are generated at the User Layer they are first queued in the transport queue in the Transport Layer. The reason for this is that the Transport Layer uses a sliding window technique to ensure that each packet arrives error-free at its destination.

The User Level of all nodes generates packets addressed to any other node with equal probability. When an optimized query schedule is simulated, nodes containing relations participating in the query, are forced to send a number of packets to specific nodes (depending on the size of the relation, the query processing tactic used and the point of execution in the schedule), depending on the particular query schedule being executed. Receiving nodes count the number of packets received. If the amount of data received equals the amount predicted in the schedule, the next segment of the schedule is executed.

Packets leaving the Transport Layer are sent to the Network Layer where a routing decision is made. The Network Layer utilizes an adaptive routing strategy which finds the shortest path between every node in the network. After making a decision as to which route is the shortest, the Network Layer queues the packet into the appropriate transmit queue (Physical Layer) for that route.

The Physical Layer has the responsibility of controlling links and of transmitting the packets one by one from the transmit queue of each link. A Medium Access Control Layer (MAC) is not required in this model, because all communicating nodes have a

dedicated link between them. In the DQDB testbed, a MAC Layer was required because all nodes shared the same link. The DQDB protocol logic ensures that only one node can access the channel at a particular point in time.

Once a packet is put onto the link, the packet arrives at the other end after a certain period of time. This transmission time is dependent on the size of the packet being transmitted and the data transmission speed of the channel. When the packet arrives, it is queued in the receive queue in the Physical Layer of the receiving node. After updating link statistics, the packet is passed to the Network Layer where a routing decision is made.

When the Network Layer of a particular node receives a packet, the packet's address is checked against the current node address. If the addresses do not match, the packet has not yet reached its destination. In this case the packet is transmitted further into the network depending on the adaptive routing algorithm. If the packet has arrived at its destination, the packet is passed to the Transport and User Layers where all relevant statistics about the packet are collected (also information related to optimized query schedules are collected). The packet is then deleted from the system.

What follows is a detailed description of the Hypercube Model.

5.2 THE HYPERCUBE TESTBED: A DETAILED DESCRIPTION

The Main Events of the Hypercube Simulation

The Hypercube model, written in SIMSCRIPT II.5, is a discrete-event simulation model. Discrete events are the atomic elements of discrete simulation. An event occurs at an

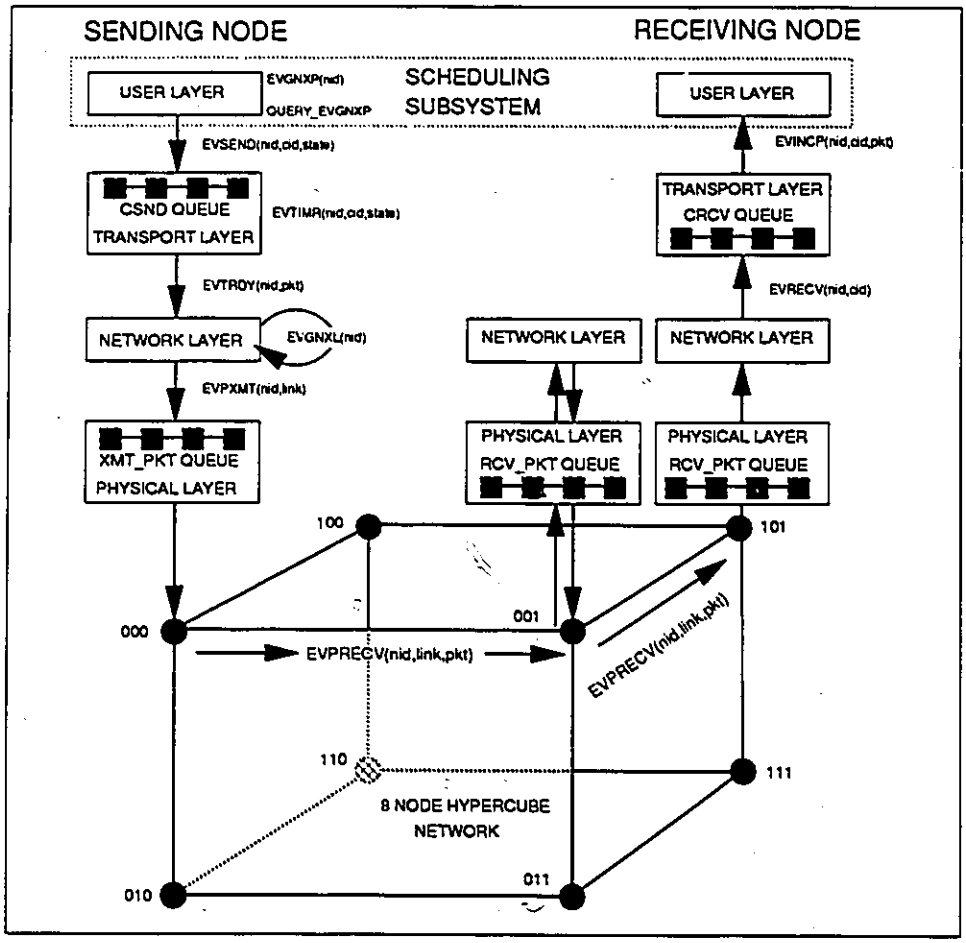


Figure 30 A Three-Dimensional Hypercube Model

instant in simulated time. Since processes have evolved from events, a process may be thought of as a sequence of events.

There are a total of sixteen events in the model. Eleven events are used to simulate the process of transmitting packets in the network (Figure 30). Table 8 describes each of these eleven events and indicates in which level of the simulated OSI model the event can be found (Figure 30). The remaining 5 are used to control the simulation program

EVENT	EXPLANATION OF EVENT	OSI LAYER
EVGNXP(nid)	EVent to Generate the NeXt Packet for this Node ID.	User
EVQURY	EVent to start execution of the distribution schedule.	User
QUERY_EVGNXP(srcid, destnid,numpkts,schedseqnum)	EVent to Generate packets relating to a part of a distributed schedule.	User
EVSEND(nid,cid,pkt)	EVent to SEND a PacKeT using the transport connection, at this Node ID, and using this Connection ID.	User
EVTRDY(nid,pkt)	EVent to Transmit this PacKeT to this Node ID.	Transport
EVTIMR(nid,cid,status)	EVent to detect an acknowledgement timeout, for this Node ID, on this Connection ID and this status for the TIMeR.	Transport
EVGNXL(nid)	EVent to GeNerate Load information packets for eXchange with neighbours of Node ID (flooding)	Network
EVPXMT(nid,link)	EVent to transmit this Packet to this Node ID using this outgoing LINK. The packet is removed from the transmit queue XMT_PKTQ for that link.	Network
EVPRCV(nid,link,pkt)	EVent signaling the arrival of a PacKeT at a Node ID, through one of its LINKs.	Physical
EVRECV(nid,cid)	EVent which signals that the network layer has decided that a packet has reached its destination Node ID on this Connection ID. This event queues the packet into the Transport's receive queue CRCVQ.	Network
EVINCP(nid,cid,pkt)	Arrival of an INComing PacKeT at the user level of Node ID, through this virtual connection.	User

Table 8 Explanation of Packet Moving Events. [13].

(Table 9).

For every send-receive pair of nodes A and B on a link, the model can be viewed

EVENT	PURPOSE
EVGREP	This EVent Generates a REPort every GR_FRQ seconds.
EVSTOP	This EVent STOPs the simulation at GSIMSTOP seconds.
EVCLKT	This EVent simulates a CLocK Tick every GCLOCKTICK seconds.
EVWMUP	This EVent signals the end of the WarM-UP period (GWARMUP seconds).
EVBTCH	This Event signals the start of BaTCH statistics collection (at GBATCH seconds).

Table 9 Events for Simulation Program Control.

as shown in Figure 30. Important implementation issues and assumptions used in the Hypercube simulation model are described below.

Simulation of the User (Application) Layer

The User Layer provides a means for application processes to access the OSI environment. This layer contains the functions necessary to support distributed applications.

The simulated User Layer is responsible for generating packets (i.e. loading the network). These packets have an exponentially distributed mean generation period and are addressed to any node in the network with equal probability.

Connection-oriented data transfer is used for exchanging data in which a logical connection is established between two communicating nodes (virtual circuit). In order to simplify the Hypercube model, the simulation assumes that each node has a logical connection to every other node in the network. This means that there are N^2—N connections (32 node Hypercube = 992 connections); no node can have a connection to itself. The destination address of a packet is used as a Connection ID.

After generating a packet, the user issues an EVSEND(nid,cid,pkt) to send this packet to its destination (Figure 30 User Layer). The Connection ID to be used is the same as the destination node [13].

In the Hypercube model, the Scheduling Subsystems of a DDMS is implemented at the user level. The Scheduling Subsystem is responsible for controlling the execution of a distribution schedule (also know in the literature as an Execution Monitor). The Scheduler coordinates the various segments in the strategy in order to present the query response at the query site.

A distributed schedule starts execution after an initial warm-up period (EVQURY). The Scheduling Subsystem assigns each segment in a schedule a sequence number. When an optimized query schedule is simulated, nodes containing relations participating in the schedule, are forced to address a number of packets to specific nodes depending on the particular query schedule being executed. When a schedule starts, the Scheduler instructs source nodes to send a certain amount of packets to a particular destination with a sequence equal to one (QUERY_EVGNXP(srcid,destnid, numpkts,schedseqnum). The Scheduler uses the distribution schedule to determine the source ID, destination ID, amount of packets to transmit and the sequence number of the schedule. The Scheduler monitors the destination node and counts the number of packets received with a sequence number equal to one. If the amount of data received equals the amount dictated in the schedule, the next segment (sequence number equal to two) of the schedule is executed. All packets not related to the query have a sequence number equal to 999. The Scheduler continually monitors and controls the sequencing of the distribution schedule until all

necessary packets arrive at the query site.

The execution time of each segment is added together to determine Total Time. The elapsed time from the start of the query until its completion is used to determine Response Time.

User (Application) Layer Assumptions

The Scheduling Subsystem of a DDMS is implemented at the User Layer. It is assumed that all information regarding coordination of schedules is globally available. Hence, no message passing is required to execute a schedule. The model does not take into account message passing and local processing time. Also, the distributed database is assumed to be non-replicated and non-fragmented. The only distributed operations that are being simulated are read only queries.

The User Layer of every node generates fixed length packets (2048 bits) addressed to any node in the network with equal probability. Packets which are addressed to the originating node (the node which created the packet) are immediately discarded.

The time between packet generation by each node is assumed to be exponentially distributed with a mean ranging from 5.0×10^{-2} second to 1.8×10^{-3} second (20 PKTs/second to 550 PKTs/second).

The transmission rate of all links is assumed to be 80Mbits/sec. The channel speed can be changed by setting the GCXMT variable from the input parameter file (see input parameter file below).

Simulation of the Transport Layer

The purpose of the Transport Layer is to provide a reliable mechanism for the exchange of data between nodes. The Transport Layer ensures that packets are delivered error-free, in sequence, with no losses or duplications.

To simulate an error-free point-to-point channel a Sliding Window protocol (IEEE 802.2 LLC's Class II) [56] is used for every connection in the network. In the model, data frames transmitted from node *A* to node *B* are intermixed with acknowledgement frames from node *B* to node *A*.

In the Hypercube model, two types of frames are used to transfer packets: they are IPDU, for information protocol data units generated at the user level (data packet); and RR for Receive-Ready frames (acknowledgements). By looking at the type field in the frame header, a receiver can tell whether the frame is data or an acknowledgment. When a data frame arrives, an acknowledgement is sent back to the sending node indicating that the packet has arrived error-free. This is known in the literature as piggybacking [57]. If there is no data packet going in the opposite direction (back to the source node), a separate RR frame is sent.

The Hypercube model assumes a Sliding Window size of one. The way this protocol works is that the sender (source node) continually transmits its data and waits for acknowledgements. Upon reception, the receiver (remote node) checks to see if the packet has any errors. If the packet arrived error-free then an acknowledgment is sent back to the source indicating that packet arrived error-free. If within a certain period of time (time-out period — see EVTIMR(nid,cid,state) in Figure 30), the receiving end

does not acknowledge receipt of the packet, the source re-transmits the packet a fixed number of times (depending on network load). The sender always keeps a copy of all unacknowledged packets. In this way, the sender is able to re-transmit packets if necessary. The sender destroys its copy of a packet upon receiving an acknowledgment for it.

The Transport Layer uses the Sliding Window protocol for every connection in the network. This means that the transport component of each node is responsible for N-1 connections. Each connection needs two connection-data-block (known in the simulation as a CENT) to keep track of details about a connection from its two ends [13].

The connection table is defined at the Transport level. The CTAB(I,J) gives the address of the connection entity (CENT), which stores all the details of the virtual connection from I to J. The peer entity CTAB(J, I) is linked to CTAB(I, J) through the CRCNC field of a CENT. Note that CTAB(I, J) = 0 for $I = J$. The connection entity is defined in Table 10.

When the Transport Layer has stored all packet information needed to maintain the connection and the Sliding Window protocol, an EVTRDY(nid, pkt) event is generated to signal the Network Layer that a packet is now ready for transmission (Figure 30).

Transport Layer Assumptions

The simulation assumes that the Transport Layer provides the User Layer with an error-free point-to-point connection. As such, every node is assumed to have a logical connection to every other node in the network. Logical connections are assumed to exist throughout the execution of the simulation.

FIELD NAME	DESCRIPTION
CRNID	Node ID at remote end of this connection.
CX	X Variable in IEEE 802.2
CLNR	Last send sequence number received from remote end.
CRCNC	Pointer to connection entity CENT at remote end.
CRETRY	Retry count (Normally 0,1,...,GN2).
CSACKT	Status of ACK - Timer : DEAD or LIVE
CSBSYT	Connection busy.
CPACKT	Pointer to ACK Timer's temporary entity.
CSTATE	State of this connection.
CRSTAT	State of remote CENT at the other end.
CVS	The send state variable.
CVR	The receive state variable.

Table 10 Data Fields Which Describes a Connection Entity (CENT)

A sliding window mechanism is used for error and flow control (sliding window size = 1). Acknowledgements are handled at this layer and piggybacking is used where applicable.

Owing to the fact that error control is handled in the Transport Layer, the model assumes that a Data Link Layer is not required.

Simulation of the Network Layer

The purpose of the Network Layer is to control the operation of the network. This includes routing of packets (packet definition given in Figure 11) through the network and controlling congestion. In broadcast networks no routing is required. As a result, the

FIELD NAME	DESCRIPTION
PSRC	The Source node ID.
PDST	The Destination node ID.
PCUR	The Current node ID.
PTIM	Packet creation time.
PQTIM	Time when the packet was put into the transmit queue (XMT_PKTQ).
PHOP	Number of hops traversed by this packet.
PNR	Send sequence number.
PNS	Receive sequence number.
PERR	Indication of error (not used in this model).
PTYP	Packet type (IPDU of RR).

Table 11 Data Fields Which Describe a Packet Entity

DQDB MAC MAN model (previously discussed) did not have a Network Layer. Owing to the fact that a Hypercube has a number of outgoing links on which to send a packet, the routing function is a major component of this model.

The Hypercube model combines two routing techniques. They are adaptive and shortest path routing. The model finds the shortest path between any two communicating nodes by making routing decisions based on recent changes in network traffic.

The simulation maintains a matrix (NDS, which is a 3-dimensional array) which represents a graph of the Hypercube network. Each node of the graph represents a node in the network. Each arc (LSET (nid)) of the graph represents a communication link. To choose a route between a given pair of nodes, Dijkstra's algorithm [69] is used to find the shortest path between them. The algorithm calculates the shortest path from the current node to every other node in the network and stores the second node of these paths in

NPATH(curr,*). In the model, arcs are labeled with the mean queueing and transmission delay in crossing the link. The algorithm uses these delays to calculate the fastest path between every two nodes. The results of these calculations are stored in NDS.

Neighboring nodes exchange estimates of link length (mean queueing and transmission delay) every 500 milliseconds. An IPDUL (an information protocol data unit containing load information) packet is sent by a source node$_i$ to every one of its neighbors. When an IPDUL packet is received, the receiver updates its load information matrix (NDS). Following this the IPDUL packet is re-transmitted on all outgoing links except for the link which it arrived on. This technique is known in the literature as flooding [56].

The method the model uses to stop incessant re-transmission of packets is for each node$_k$ to keep track of the last packet sequence number generated by every other node$_i$ in the network. If an IPDUL packet is received with a sequence number equal to or less than the last sequence number received from node$_i$, the packet is discarded. Also, to help with the clean-up of very old IPDUL packets, there is a maximum length of time that these packets can exist in the network.

The matrix NDS(K, I,J) gives a measure of distance for the outgoing edges, J = 1, 2, ... , NUMBER.OF.LINKS(I), and I = 1, 2, ..., NUMBER.OF.NODES. The index K identifies the node at which the distance matrix (generated through I and J) is being used. As discussed above, adaptive routing maintains a distance matrix for every node in its own area. The last two indices of NDS-array refer to the distance matrix entries while the first index refers to the node which owns the distance matrix. An example of the distance matrix for node 1 in a three-dimensional Hypercube, is given in Table 12.

DISTANCE MATRIX AT NODE 1.	THE DELAY ON EACH OUTGOING LINK					
DISTANCE FROM NODE *I*	DELAY (millisec)	TO NODE *J*	DELAY (millisec)	TO NODE *J*	DELAY (millisec)	TO NODE *J*
1	0.224	2	0.255	3	0.211	5
2	1.000	1	0.274	4	0.241	6
3	1.000	1	0.239	4	0.254	7
4	0.222	2	0.377	3	0.233	8
5	1.000	1	0.229	6	0.226	7
6	0.241	2	0.297	5	0.229	8
7	0.232	3	0.229	5	0.289	8
8	0.274	4	0.317	6	0.213	7

Table 12 An Example of the Distance Matrix (NDS(1,i,j)) at Node 1.

The simulation uses Dijkstra's algorithm on matrix NDS(*k,i,j*) (Table 12) to calculate the main routing table NPATH(*I,J*). The *I*-th row of NPATH holds the GN-1 (total number of nodes–1) different path details (excluding the path from *I* to *I*). Actually only the second node of these paths are maintained in NPATH(*I,J*), the first node being the starting node *I*. An example of the NPATH matrix is given in Table 13. If node 1 wants to send a packet to node 8 (NPATH(1,8)) the shortest path is: node 1 — node 5 — node 6 — node 8 (Table 13).

Network Layer Assumptions

This layer assumes the Datagram model. In the Datagram model, each packet is routed independently through the network. Successive packets may follow different routes and

	DESTINATION NODE *J*							
SOURCE NODE *I*	1	2	3	4	5	6	7	8
1	0	2	3	3	5	5	5	5
2	1	0	1	4	1	6	1	6
3	1	1	0	4	1	1	7	4
4	2	2	3	0	2	2	8	8
5	1	1	7	7	0	6	7	6
6	2	2	8	8	5	0	8	8
7	3	3	3	3	5	5	0	8
8	7	4	7	4	7	6	7	0

Table 13 An Example of the Shortest Path Routing Table (NPATH) for a Three-Dimensional Hypercube Network.

hence, may arrive out-of-order. Datagram networks are more robust and adapt to failures and congestion more easily than connection-oriented networks.

The shortest path adaptive routing algorithm is used which bases its routing decision on measurements or estimates of current traffic and topology.

Flooding is used to communicate link load information to all nodes in the network.

No flow control mechanism is used. Infinite buffer space is assumed at the receiving end of each link.

Simulation of the Physical Layer

The physical link is mainly concerned with the transmission of bits over a communication link.

When the Network Layer of a node wants to send a packet using one of its links, it queues up the packet in the transmit queue (XMT_PKTQ) of that link and issues an

EVPXMT(nid,link). The Physical Layer removes the packet from the XMT_PKTQ FIFO queue and schedules a receive event EVPREC for the node at the receiving end (Figure 30). The time the EVPREC event is scheduled to occur is based on the packet size and the transmission speed of the link.

Once a $link_i$ has started transmitting, it cannot transmit another packet until the current transmission is complete. Another event is scheduled for itself (equal to the transmit time for this packet) at a later time to see if another packet has been queued for transmission over $link_i$.

After receiving a packet through an incoming link the Physical Layer checks the packet's destination field (PDST) against the current node ID. If they do not match, the routing table (NPATH) is used to route the packet further into the network. For packets having reached their destination, an event EVRECV is generated to signal the Transport Layer of the arrival of a packet (Figure 30).

A node in general has a number of outgoing edges. In the following arrays this information is maintained.

NSZ(*I*) gives the number of outgoing edges for the *I* th node.

The node-IDs at the receiving end of these outgoing edges for node *I* are NIX(*I*,1), NIX(*I*,2), ... , NIX(*I*, NSZ(*I*)). Thus, the NIX-array actually stores the indices of the outgoing neighbors of a node.

Physical Layer Assumptions

All nodes are assumed to be homogeneous. The physical links which interconnect nodes

are assumed to employ Wavelength Division Multiplexing. All links are assumed to be unidirectional. The speed, capacity and length of all communication links are identical (80Mbits/sec).

It is assumed that there is no limit to the buffering capacity of the queues.

It is also assumed that all packets arrive at a remote node error free and that links never fail.

Input Parameters for the Hypercube Model

The model is set up to generate and simulate any size Hypercube store-and-forward network by reading the input file "hcube.dat". What follows is a description of the variables assigned values from this file.

Network Description

GNET: Indicates the type of network to simulate. GNET = 1 indicates that a Hypercube network will be simulated.

GN: Indicates the number of nodes to simulate. GN = 32 indicated that the Hypercube will have 32 nodes.

User Level

GIMD: Exponentially distributed inter-packet generation time. GIMD = 0.0125 indicates that each node will generate 80 PKTs/sec.

Physical Level

GCXMT: Channel transfer rate. GCXMT = 0.0000001 indicates a channel speed of 80Mbits/sec.

GMOVHD: Overhead time to transfer one packet. GMOVHD = 0.000250 indicates a 250 microsecond overhead.

Network Level

GLDSTRT: Time to start exchanging load (IPDUL) packets. GLDSTRT = 0.004 indicates that the exchange of IPDUL packets is to start at 0.04 second.

GLDEXCG: Frequency of load-info generation (i.e., Flooding). GLDEXCG = 0.50 indicates that load information packets must be exchanged every half second.

GMXLAGE: Maximum age of any IPDUL packet on the network. GMXLAGE = 0.40 indicates that the packet should be destroyed if it has been in the network longer than 0.40 second.

GFERR: Allowable fraction of packets with errors against total number of packets on a LINK. GFERR = 0.01 indicates that one percent of the packets will have errors.

Transport Level

GK: Sliding window size. GK = 1 indicates a sliding window count of one.

GN2: Retry count. Maximum number of times to resend a packet.

GDBPT: Acknowledgment time-out period. This is described as a function of the send queue length at the Transport Layer.

Simulation Run Control

GSIMSTOP: Run length of the simulation.

GCLOCKTICK: Period to display messages and time on the terminal.

GWARMUP: Signals the end of the warm-up period.

GBATCH: Starts scheduling EVBTCH for batch statistics collection.

Fault Model

The fault model is not used in the testbed. All fault model variables are set so their effects are negligible.

5.3 MODEL VALIDATION AND VERIFICATION

An attempt to validate and verify the original Supercube model is documented in [13]. In [13], it is stated that as a result of not having real life data available for model validation, expert opinion and user input were used throughout the development phase of the project. All data requirements for the model were met by assuming hypothetical figures after numerous consultations with experts and the model users. The verification task was performed by:

1. dumping the network connectivity graph.
2. dumping vital data structures like PKT entities, CENT entities, NPATH entities and NDS entities.
3. tracing every possible communication between two connection entities over a virtual connection.
4. performing a complete trace of SIMSCRIPT's dynamic memory usage.

Following this, a number of cross-checks were performed from numerous reports generated by the program. [13] states that "the possibility of an improperly functioning program for the simulation model is almost eliminated through these cross-checks".

QUERY PATH (TRANSMITTED SERIALLY)	AMOUNT OF DATA TRANSMITTED(PACKETS)	HAMMING DISTANCE	PKT HOPS
R1 - R2	100	1	100
R2 - R3	16	3	48
R3 - QS	9	5	45
		TOTAL PACKET HOPS =	193

Table 14 Sample Query for Validation Purposes

However, [13] did not prove this claim and therefore this statement should be treated with some skepticism.

The same procedure of dumping, tracing and cross-checking was performed on the Hypercube model.

Table 14 shows a sample query with the amount of data transmitted between each relation. Also depicted in Table 14, is the distance (measured in this case in hops) each packet has to travel to reach its destination. The total number hops to execute the query is 193.

In the Hypercube model, the time to transmit one character is 0.1 microseconds. Each packet contains 2048 characters. Hence, the time to transmit a packet is 0.2048 milliseconds. The time to transmit a packet with an acknowledgment is 0.4096 milliseconds (0.0002048 X 2). The theoretical time to execute the query path in Table 14 at no load on the network is 0.079 seconds (0.0004096 X 193). This compares very well with the actual simulated time of 0.082 seconds.

With each node on the network generating 350 packets per second the average time to transmit a packet (with acknowledgment) is 0.729 milliseconds. The theoretical time

to execute the query path in Table 14 at a load of 350 packets per second is 0.140 seconds (0.000729 X 193). This compares very well with the actual simulated time of 0.143 seconds.

5.4 EXPERIMENTING WITH OPTIMIZED DISTRIBUTED QUERIES

As discussed in Chapter 2, the Query Processing Subsystem of a DDMS is responsible for implementing the query optimization algorithm (i.e. PARALLEL, SERIAL, and GENERAL). The purpose of the query optimization algorithm is to generate an efficient distribution strategy for a particular query. This distribution strategy is then passed to the Scheduling Subsystem for execution. The Scheduling Subsystem sends commands to local DBMSs in the network in order to do local data processing related to the query. The Scheduler also interacts with the network communications facilities of each node (nodes involved with the query) in order to transmit data as determined by the distribution strategy. The Scheduler coordinates the various schedules in the strategy so that the query response is presented at the result node. At this point in time, the Query Processing Subsystem is not implemented in the model. Execution of query optimization algorithms (i.e. PARALLEL, SERIAL, GENERAL and JOIN) are performed outside the simulation (mechanically-by hand).

The User Level is responsible for creating packets. The User Level of each node generates packets addressed to other node with equal probability. When the Scheduling Subsystem executes a distribution strategy, nodes containing relations participating in the query, are forced to address a number of packets to specific nodes, depending on the

particular distribution strategy being executed. To simplify the model, it is assumed that the Scheduling Subsystem has direct control over all nodes. As a result of this assumption, control messages needed to coordinate the execution of the distribution schedule are not simulated. Receiving nodes count the number of query related packets received. If the amount of data received equals the amount dictated in the schedule, the Scheduling Subsystem coordinates the next segment of the distribution schedule.

A simple[6] query example is presented in Figure 31 (Example 1) to generate distribution schedules to test the Hypercube testbed. The three distribution strategies generated in Example 1 are: the Initial Feasible Solution strategy, the Algorithm PARALLEL distribution strategy, and the Algorithm SERIAL distribution strategy. It is these distribution schedules that are implemented in the simulation.

Figure 31 (Example 1) illustrates a materialization for a simple query. This example assumes that R_1, R_2, R_3 and R_4 (listed in ascending order) are all required at the query site to answer the query. The location of each relation in a 32 node Hypercube network is also depicted in Figure 31. Relation R_1 is located at node 0, relation R_2 is located at node 7, relation R_3 is located at node 15 and relation R_4 is located at node 23. The query site is located at node 31.

The first processing tactic employed is to send all relations directly to the query site. This tactic is known in the literature as the Initial Feasible Solution (IFS). The IFS for the query is depicted in Figure 31 with the response time (RT) and total time (TT) being 1020 and 2280 units respectively.

[6]The word simple indicates that only one attribute from each relation is involved in the query.

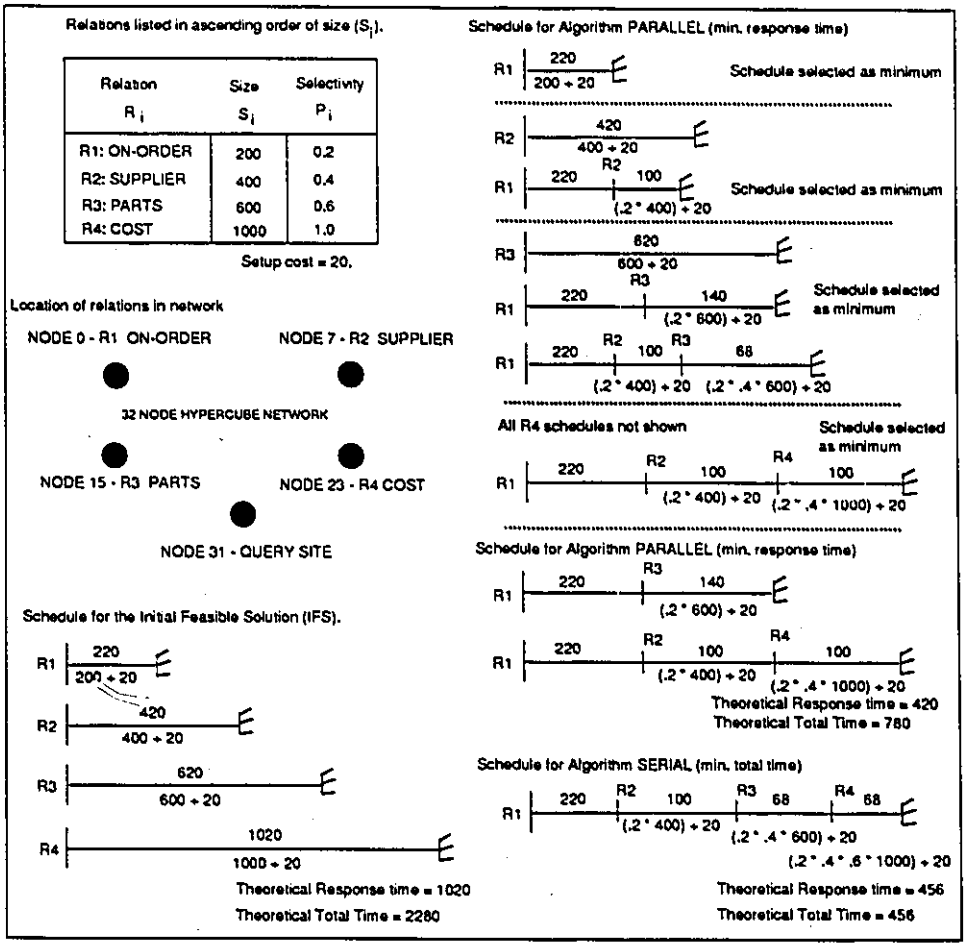


Figure 31 Example 1 — Optimized Distribution Schedules for a Simple Query.

Algorithm PARALLEL (see Chapter 2 for details) is applied to the materialization to generate a minimized response time schedule for the query. A summary of the most cost beneficial schedules for each relation is given in Figure 31. The integrated distribution schedule for Algorithm PARALLEL is shown with the response time and total time being 420 and 780 units respectively. Executing Algorithm PARALLEL statically (outside the simulation) produces a distribution schedule which is expected to reduce response time by 59 percent when compared to the IFS (simulation results are discussed in the next

section).

Algorithm SERIAL (also see Chapter 2) is applied to the materialization to generate a minimized total time schedule for the query. The integrated distribution schedule for the query using Algorithm SERIAL is shown with the response time and total time being 456 and 456 units respectively. Executing Algorithm SERIAL statically produces a distribution schedule which is expected to reduce total time by 80 percent when compared to the IFS.

While the optimization algorithms presented above work for special situations, they do not necessarily derive efficient distribution strategies for general queries. A general[7] query example is presented in Figure 32 (Example 2) to generate more complicated distribution schedules to test the Hypercube testbed. The three general distribution strategies generated in Example 2 are: the IFS strategy, the Algorithm GENERAL distribution strategy, and a serial JOIN distribution strategy. It is these resulting distribution schedules that are implemented in the simulation.

Figure 32 (Example 2) provides a summary of the distributed query optimization example given in Chapter 2. Figure 32 illustrates a materialization for the general query outlined in Chapter 2. Example 2 assumes that R_1, R_2, and R_3 (listed in ascending order) are required at the query site in order to answer the query. The location of each relation in a 32 node Hypercube network is also depicted in Figure 32. Relation R_1 is located at node 0, relation R_2 is located at node 15, and relation R_3 is located at node 23. The

[7]In this context, the word general indicates that more than one attribute from each relation can be involved in the query.

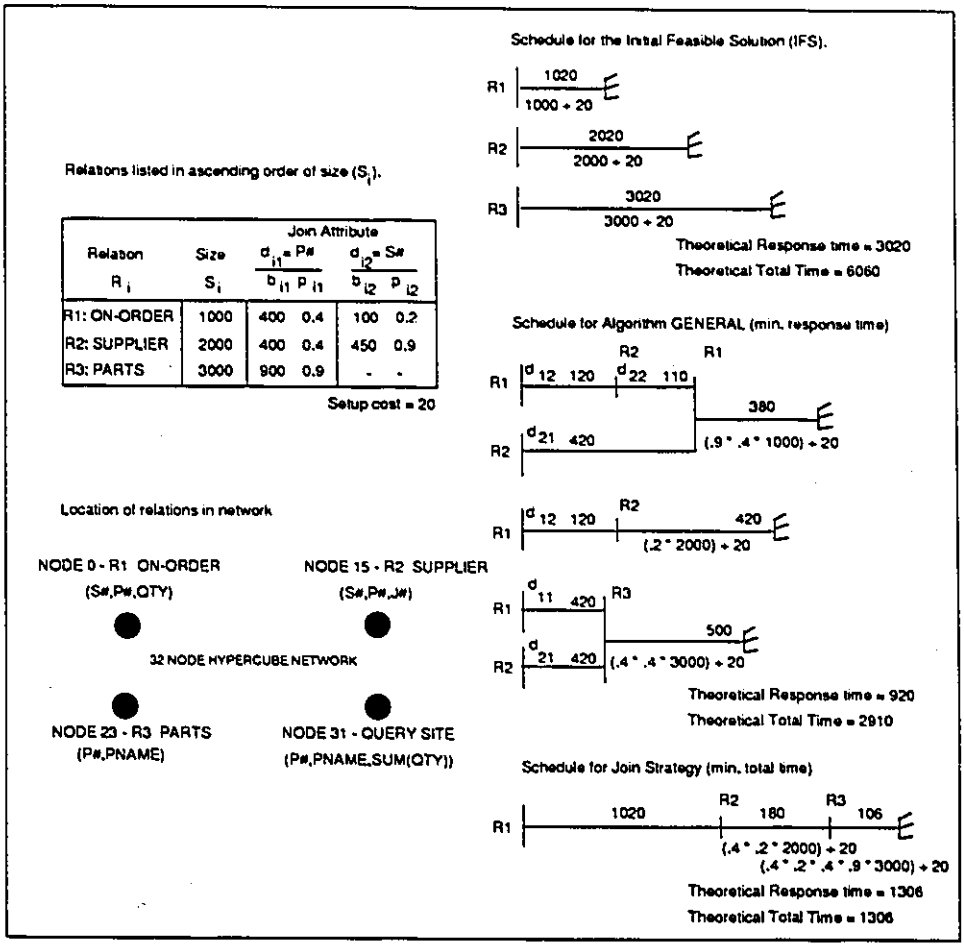


Figure 32 Example 2 — Optimized Distribution Schedules for a General Query.

query site is located at node 31. The IFS for the query is depicted in Figure 32 with the response time (RT) and total time (TT) being 3020 and 6060 units respectively.

Algorithm GENERAL is applied to the materialization to generate a minimized response time schedule for the query. The integrated distribution schedule for Algorithm GENERAL is shown with the response time and total time being 920 and 2910 units respectively. Executing Algorithm GENERAL statically produces a distribution schedule

which is expected to reduce response time by 70 percent when compared to the IFS.

The JOIN tactic (see Chapter 2) is applied to the materialization to generate a minimized total time schedule for the query. Algorithm PARALLEL, SERIAL and GENERAL optimize queries based on the SEMI-JOIN tactic. The trade-offs between using JOIN and SEMI-JOIN as a query processing tactic are discussed in Chapter 2. The integrated distribution schedule for the query using the JOIN strategy is shown with the response time and total time being 1306 and 1306 units respectively. Executing the JOIN algorithm statically (outside the simulation) produces a distribution schedule which is expected to reduce total time by 80 percent when compared to the IFS TT (simulation results are discussed in the next section).

5.5 RESULTS AND CONCLUSIONS OF THE HYPERCUBE TESTBED INVESTIGATION

Total Time Minimization Using Algorithm PARALLEL and SERIAL (Experiment #1)

Figure 33A depicts the results of increasing load on the *total time* it takes to execute the IFS TT schedule, the Algorithm PARALLEL TT schedule and the Algorithm SERIAL TT schedule for the example simple query given in Figure 31 (Example 1). Figure 33A shows that the IFS TT is more sensitive to network load than the other total time schedules. The Algorithm PARALLEL TT schedule shows approximately a 55 percent reduction in total time (theoretical results discussed in the previous section predict a 59 percent reduction).

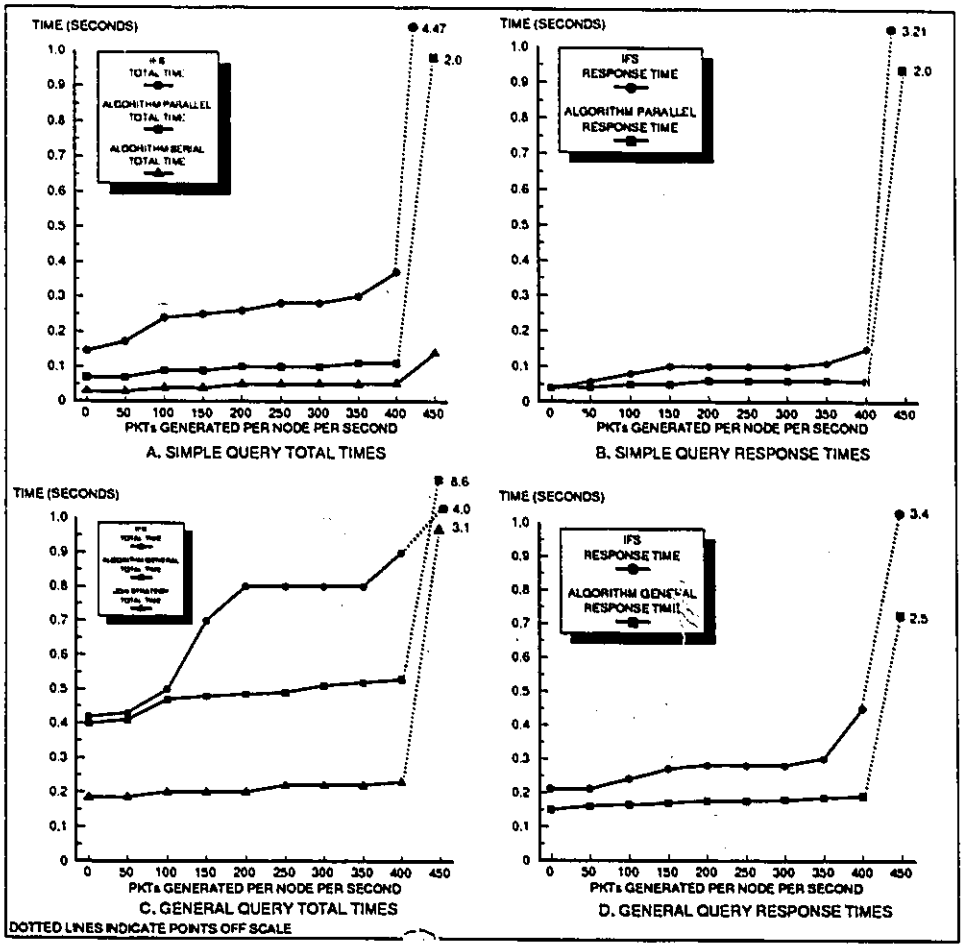


Figure 33 Results of Experimenting with Simple and General Queries (Example 1 & 2).

When the network reaches its overload region (450 packets per second) the IFS and Algorithm PARALLEL TT schedules show an exponential increase in total time. A possible explanation is that queue length, and hence queueing delay, has grown large because of the increased load. As a result, total query time increases exponentially because this increased delay is encountered at every hop in the query path.

The Algorithm SERIAL TT distribution schedule shows the most reduction in total

time. The reason is that Algorithm SERIAL has joined smaller relation to larger relations thus reducing the amount of data transferred. With less data being transmitted in the network, the effect of queueing delay is reduced. Owing to this fact, total query time experiences less sensitivity to network load.

Figure 33A shows that the Algorithm SERIAL TT schedule is the least sensitive to network load. The total time needed to execute the query is reduced by 80 percent when compared to the IFS TT strategy (theoretical results predict an 80 percent reduction). It also shows that the total time minimization strategy performs significantly better when the network becomes overloaded. This result coincides with what is predicted in the literature [31].

Response Time Minimization Using Algorithm PARALLEL and SERIAL (Experiment #1)

Figure 33B depicts the results of increasing load on the *response time* (elapsed time) it takes to execute the IFS RT schedule and the Algorithm PARALLEL RT schedule. The Algorithm SERIAL RT schedule is shown in Figure 33A, since total time is equal to response time for a serial schedule.

Figure 33B shows that the Algorithm PARALLEL response time distribution schedule takes less time to execute than the IFS RT schedule. The reason for this is that Algorithm PARALLEL has reduced the amount of data transmitted in the network with the use of the SEMI-JOIN query processing tactic.

Figure 33B shows that Algorithm PARALLEL has succeeded in reducing response time by approximately 40 percent (theoretical results predict a 60 percent reduction).

A possible explanation for the discrepancy is that although Algorithm PARALLEL succeeded in reducing the amount of data transmitted, it has not significantly decreased the total number of packet hops[8] required to present the data at the query site. The longest parallel transmission in the IFS schedule, R_1 to the query site, uses 100 packet hops to send its data to the query site. The longest parallel transmission in the Algorithm PARALLEL distribution schedule, R_1 to R_3 to the query site, uses 90 packet hops to transmit its data to the query site.

Figure 33B also shows response time of the IFS schedule and the Algorithm PARALLEL schedule increasing exponentially when the network becomes overloaded (450 packets per second). A possible explanation for this is that queueing delay has grown large because of the increased load. As a result, total query processing time increases exponentially because this increased delay is encountered at every hop in the query path.

When Figure 33B is compared to the result of the Algorithm SERIAL schedule in Figure 33A (total time equals response time for serial strategies), the SERIAL schedule shows approximately the same response time as the PARALLEL schedule in Figure 33B. This result does not coincide with the theory. The explanation for this is that although the query path of the Algorithm SERIAL distribution strategy (R_1–R_2–R_3–R_4–QS) traverses 7 hops[9] and the Algorithm PARALLEL distribution strategy (R_1–R_3–QS) traverses a total of 5 hops, Algorithm SERIAL has succeeded in reducing the amount of data

[8]A packet hop is a unit of measure that describes one packet traversing one link. Thus 2 packets traversing 3 links (hamming distance = 3) is equal to 6 packet hops.

[9]The number of hops is a theoretical value based on the sum of all hamming distances between R_1 and the Query Site.)

that must traverse these two extra hops. In the Algorithm PARALLEL distribution strategy (R_1–R_3–QS) sends 200 packets from R_1 to R_3 over 4 hops while the Algorithm SERIAL distribution strategy (R_1–R_2–R_3–R_4–QS) sends 200 packets from R_1 to R_2 over 3 hops. The Algorithm PARALLEL distribution strategy (R_1–R_3–QS) actually cost the same as the Algorithm SERIAL distribution strategy (R_1–R_2–R_3–R_4–QS). Without using the knowledge of network topology and relation location, Algorithm PARALLEL was unsuccessful in reducing response time (below that of the SERIAL Strategy) in the simulation.

Another important factor is that in packet switched networks which use algorithms such as the Shortest-Path-Algorithm to route packets, it is impossible to predict beforehand what path a packet will actually take to reach its destination. It seems likely that query optimization can be improved by making use of the information available in the network routing table (queueing delay) to determine the true cost of a strategy. Lastly, it seems likely that knowledge of network topology and where relations are located in relation to one another (i.e. number of hops apart) would also provide valuable information for query optimization algorithms. Detailed analytical and simulation studies are required to determine if these speculations are true.

Total Time Minimization Using Algorithm GENERAL and the JOIN Strategy (Experiment #1)

Figure 33C depicts the results of increasing load on the *total time* needed to execute the IFS TT schedule, the Algorithm GENERAL TT schedule and the JOIN Algorithm strategy schedule for the general query given in Figure 32 (results from Example 2).

Figure 33C shows that the IFS TT is more sensitive to network load than the other total time schedules. The query path of the IFS distribution strategy (R_1–QS) transmits 100 packets over 5 hops. This causes 100 packets to experience long delays at five nodes as load increases.

The Algorithm GENERAL TT schedule is less sensitive to changes in network load (between 100 and 400 packets per second) than the IFS TT schedule. It has also successfully reduced the total time needed to execute the query. The reason is that R_1, which is 5 hops away from the query site, has had its query related data reduced from 100 packets to 36 packets by Algorithm GENERAL (SEMI-JOIN strategy). This reduces the packet hops required to execute this segment of the schedule from 500 to 180. The cost of performing the SEMI-JOIN is 36 packet hops. Thus the total cost savings is 284 packets hops (500 — 216).

However, when the network becomes overloaded (450 packets per second), the Algorithm GENERAL TT schedule actually takes more time than the IFS TT schedule. A possible explanation for this is that node location and queueing delay has not been taken into account when the query was optimized. In example 2, the hamming distance between R_1 and the query site is 5. When the network becomes overloaded, a packet experiences long queueing delay for every hop. When executing the SEMI-JOIN for R_1 in the Algorithm GENERAL TT schedule (R_1–R_2–QS) packet traverse a total 13 hops (theoretical value based on hamming distance). Each packet experiences longer delays at each hop as network load increases. Although the amount of data transferred in the IFS distribution schedule is higher (R_1 — QS, = 500 packets), the data traverses only

one link (hop).

The JOIN algorithm TT distribution schedule (Figure 33C) shows the most reduction in total time. The total time needed to execute the query is reduced by 50 to 75 percent in the underload region (theoretical results predict an 80 percent reduction) and approximately 22 percent in the overload region, when compared to the IFS TT strategy. Figure 33C shows that the JOIN TT schedule is the least sensitive to network load. The reason for this is that the JOIN Strategy has reduced R_2 from 40 packets to 16 packets and R_3 from 48 packets to 9 packets when compared to the Algorithm GENERAL distribution schedule. It also shows that the JOIN total time minimization strategy performs significantly better than Algorithm GENERAL TT schedule when the network becomes overloaded. It is by coincidence that by following this serial strategy, the number of hops (delay) is less than the Algorithm GENERAL distribution strategy. This graph supports the speculation that by minimizing the number of hops in a query schedule, query execution can be further reduced.

Response Time Minimization Using Algorithm GENERAL and the JOIN Strategy (Experiment #1)

Figure 33D depicts the results of increasing load on the *response time* (elapsed time) that it takes to execute the IFS RT schedule and the Algorithm GENERAL RT schedule. The JOIN algorithm RT schedule is shown in Figure 33C, since total time is equal to response time for a serial schedule. Figure 33D shows that the Algorithm GENERAL response time distribution schedule is less sensitive to network load than the IFS RT schedule. It shows that Algorithm GENERAL has succeeded in reducing response time by approximately

36 percent (theoretical results predict a 70 percent reduction). A possible explanation for this discrepancy is that Algorithm GENERAL was executed statically (outside the model) without taking dynamic factors such as network topology, load and relation location into account. The theoretical IFS Response Time (Figure 32) assumes R_1 and R_3 are the same distance from the query site. Owing to the fact that R_3 is the largest relation, it has the most affect on response time. In this experiment, R_3 is one hop from the query site while R_1 is 5. Also, Algorithm GENERAL produces a parallel strategy which first transmits 10 packets to R_2 (3 hops), then R_2 transmits 9 packets back to R_1 (3 hops) and finally R_1 send 36 packets to the query site (5 hops). By not taking network topology, load and relation location into account, the theoretical results predicted by Algorithm GENERAL became inflated.

Figure 33D also shows that the response time of the IFS schedule and the Algorithm GENERAL schedule increases exponentially when the network becomes overloaded (450 packets per second).

When Figure 33D is compared to the result of the JOIN algorithm in Figure 33C, the GENERAL RT schedule shows a reduction in response time when compared to the JOIN (TT & RT) schedule in Figure 33C. This behavior is predicted in the literature [32].

It is stated in [31] that "executing parallel schedules requires more coordination and hence more message passing than a serial strategy". At the present time, message passing time is not considered in the results. It seems possible that if message passing time was taken into account, the time advantage that the Algorithm GENERAL RT schedule has over the JOIN schedule would be reduced or even negated.

It is unclear whether it is a waste of time and effort to develop a minimized RT schedule. [54] argues that increasing the overhead in forming complicated optimal strategies is only marginally beneficial. This statement supports the Hypercube Testbed results. Parallel strategies take longer to calculate and require more synchronization versus serial strategies that are simpler to calculate and require less synchronization. The following paragraphs elaborate on this point.

[54] states that because of many parallel data transmissions to an intermediate node, the resulting joined relation cannot be processed or transmitted before all attributes have arrived. It is stated that this synchronization (waiting for the last attribute — the transmission with the most delay) could lead to an increase in response time. Owing to the fact the R_1 and R_3 each wait for parallel transmissions (Figure 32 — Algorithm GENERAL), processing must wait for the slowest transmission to complete. Thus, the probability that response time will be delayed, increases as the number of parallel transmissions required to perform a SEMI-JOIN or JOIN at a particular site increases.

[54] also states that "minimizing response time leads to an increased number of parallel data transmissions in the query processing strategy. In multiprocessor systems, under moderate to heavy load, these extra transmissions may lead to significant queueing delays and synchronization delays; delays which may cause poor query response time. By minimizing the total time in a query processing strategy, fewer transmissions will be included, and improved actual response times may result". Simulation results from both examples show this to be the case. [31] also states that serial strategies are more efficient to process.

The relatively short schedules execution time exhibited in Figure 33D tended to support the statement that local processing cost should be considered when optimizing queries in fast networks. [70] states that with fast networks, local processing cost is as important as the communication cost and sometimes even more important. In this model, local processing cost is not taken into account. Figure 33C shows that JOIN query processing tactic performs reasonably well compared to the SEMI-JOIN query processing tactic (Algorithm GENERAL distribution schedule) shown in Figure 33D. It is stated in [71] that JOINs requires less *local* processing time to compute than SEMI-JOINS.

Total Time Minimization Using Algorithm PARALLEL and SERIAL (Experiment #2)

Although the algorithms succeeded in reducing both the total time and response times in the simulation, knowledge of relation location, query site and queueing delays could possibly produce different schedules which might reduce these times even further. The algorithms discussed above do not consider these dynamic system factors. To explore this notion further, a second set of experiments was conducted.

Table 15 shows the changes in node location for a second set of experiments. The second set of experiments executes the same distribution strategies on the same network under the same loading conditions. The only change is the location of the relations on the network. Table 15 shows the relations listed in ascending order of size and the changes in hamming distance between the first set of experiments and the second. The second set of experiments is setup so that the hamming distance between relations (listed in ascending order of size) has increased.

EXAMPLE ONE					
FIRST SET OF EXPERIMENTS			SECOND SET OF EXPERIMENTS		
RELATION	NODE #	LOCATION	RELATION	NODE #	LOCATION
R1	0	00000	R1	0	00000
R2	7	00111	R2	15	01111
R3	15	01111	R3	16	10000
R4	23	10111	R4	23	10111
QS	31	11111	QS	8	01000
EXAMPLE TWO					
FIRST SET OF EXPERIMENTS			SECOND SET OF EXPERIMENTS		
RELATION	NODE #	LOCATION	RELATION	NODE #	LOCATION
R1	0	00000	R1	0	00000
R2	15	01111	R2	16	10000
R3	23	10111	R3	23	10111
QS	31	11111	QS	8	01000

Table 15 Changes in Node Location for Second Set of Experiments

Figure 34A depicts the results of increasing load on the *total time* it takes to execute the IFS TT schedule, the Algorithm PARALLEL TT schedule and the Algorithm SERIAL TT schedule (with the new relation locations give in Table 15) for the example simple query given in Figure 31 (Example 1).

Response Time Minimization Using Algorithm PARALLEL and SERIAL (Experiment #2)

Figure 34B depicts the results of increasing load on the *response time* (elapsed time) it takes to execute the IFS RT schedule and the Algorithm PARALLEL RT schedule with

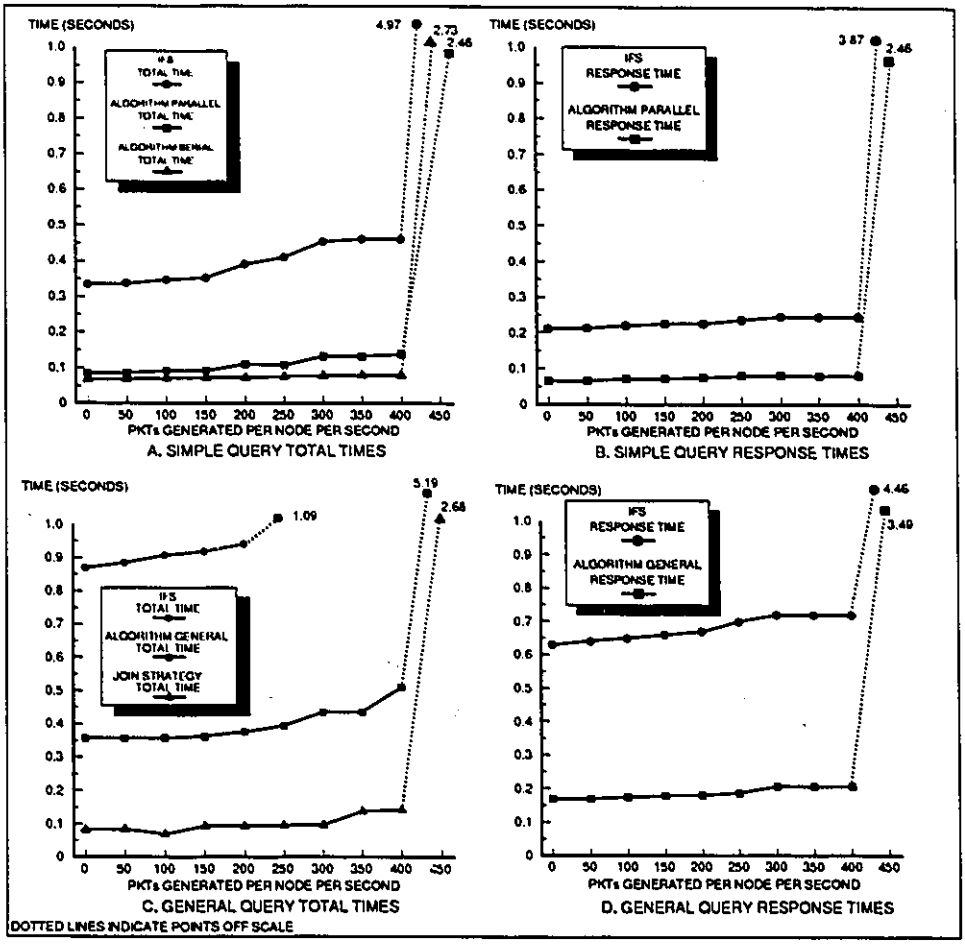


Figure 34 Results of Experimenting with Example 1 & 2 with Different Relation Locations.

the new relation locations given in Table 15. The Algorithm SERIAL RT schedule is shown in Figure 34A, since total time is equal to response time for a serial schedule.

When the distribution schedules in Figure 34A and Figure 34B are compared to their counter parts in Figure 33A and Figure 33B they show an increase in execution time. An increase in total time and response time is to be expected (Example 1) because of the increase in distance (hamming distance) between relation involved in the query (see

EXAMPLE ONE											
FIRST SET OF EXPERIMENTS						SECOND SET OF EXPERIMENTS					
IFS		ALGO. PARA.		ALGO. SERIAL		IFS		ALGO. PARA		ALGO. SERIAL	
TT	RT	TT	RT	TT	RT	TT	RT	TT	RT	TT	RT
340	100	170	70	83	83	900	500	180	128	135	135
EXAMPLE TWO											
FIRST SET OF EXPERIMENTS						SECOND SET OF EXPERIMENTS					
IFS		ALGO. GEN.		ALGO. JOIN.		IFS		ALGO. GEN.		ALGO. JOIN.	
TT	RT	TT	RT	TT	RT	TT	RT	TT	RT	TT	RT
1000	500	800	360	424	424	2000	1500	705	400	188	188

Table 16 Total Time and Response Time Measured in Packet Hops for Experiment 1 & 2.

Table 15 Example One).

When the Algorithm SERIAL distribution schedule of Figure 34A is compared to the Algorithm SERIAL distribution schedule of Figure 33A it shows a large increase in query execution time in the overload region (450 packets per second). The reason for this is that the hamming distance has tripled at the end of the query path (R_2 — R_3 — QS increased from 4 to 13). Owing to this fact and the fact that queueing delay is increasing exponentially, an exponential increase in query execution time results.

When Figure 34B is compared to the result of the Algorithm SERIAL RT schedule in Figure 34A (total time equals response time for serial strategies), the Algorithm GENERAL RT distribution schedule shows a reduction in response time. In this case the result coincides with the theory. This result shows that node location and network topology does have an effect on query execution time.

Total Time Minimization Using Algorithm GENERAL and the JOIN Strategy (Experiment #2)

When the IFS distribution schedules in Figure 34C and Figure 34D are compared to their counter parts in Figure 33C and Figure 33D they show an increase in execution time. An increase in IFS total time and response time is to be expected (Example 2) because of the increase in distance (hamming distance) between relations involved in the query (see Table 15 Example Two, IFS, TT & RT).

When the Algorithm GENERAL TT distribution schedule in Figure 34C is compared to Algorithm GENERAL TT distribution schedule in Figure 33C it shows a decrease in total time. A decrease in total time is to be expected (Example 2) because of the decrease in distance (hamming distance) between relations involved in the query (see Table 15Example Two, Algo. GEN., TT).

Response Time Minimization Using Algorithm GENERAL and the JOIN Strategy (Experiment #2)

When the Algorithm GENERAL RT distribution schedule in Figure 34D is compared to the Algorithm GENERAL RT distribution schedule in Figure 33D it shows an increase in response time. An increase in response time is to be expected (Example 2) because of the increase in distance (hamming distance) between relations involved in the query (see Table 15, Example Two, Algo. GEN., RT).

When the Algorithm JOIN TT distribution schedule in Figure 34C is compared to the Algorithm JOIN TT distribution schedule in Figure 33C it shows a decrease in total time. An decrease in total time is to be expected (Example 2) because of the decrease

in distance (hamming distance) between relations involved in the query (see Table 15, Example Two, Algo. JOIN, TT).

The JOIN Strategy depicted in Figure 34C shows the lowest total time and response time (total time equals response time for serial strategies) for all query schedules in Example 2. The reason for this is that this schedule is the minimal schedule based on these relations and the locations given in Table 16

Conclusion

It seems likely that minimizing the number of hops in a query schedule would be beneficial in query optimization. However, without reading the network routing table it is impossible to know exactly how many hops a packet will actually traverse, and the queueing delay it will experience as it travels through the network. Algorithms which utilize the information in the network routing table should be developed and studied.

By constructing the Hypercube Testbed and by implementing and experimenting with the above examples, a user of the Hypercube testbed can:

- ☐ define different network topologies on which a Distributed Database System is stored.
- ☐ experiment with different heuristics for query optimization.
- ☐ define traffic and other dynamic characteristics of the network at a given point in time.
- ☐ simulate different network sizes.
- ☐ start queries at a specific point in time.

- ☐ have a fixed or random number of relations.
- ☐ have fixed or random relations sizes.
- ☐ change the location of relations.
- ☐ make the location of the query site fixed or random.
- ☐ force nodes (CPUs) to wait, based upon the query processing tactic being simulated.

In the future, algorithms incorporating knowledge of network topology and network load to optimize queries will be developed and tested on the testbed. The results of these "extended" algorithms will be compared to the results of existing algorithms.

CHAPTER 6

SUMMARY OF CONCLUSIONS

CONTRIBUTIONS OF THE THESIS WORK AND COMPLETED OBJECTIVES

The first major contribution of this Thesis work involved the development of a Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN) simulation model using the NETWORK II.5 simulation package. This testbed is available to future students who wish to continue the investigation into the DQDB MAC MAN protocol.

The development of the DQDB MAN model included the independent development of detailed DQDB Medium Access Control (MAC) Protocol logic. This logic is documented in [30].

Sixteen different network description programs were developed to perform an investigation into different DQDB MAC MAN architectures. Over 150 experiments were conducted to determine:

- Average packet queueing time
- Average slot queueing time
- Average queue length

- Throughput network capacity
- Max operating range
- Slot utilization
- Queueing time by location on network

Results of these experiments and performance analysis were first documented in [72].

The second contribution of this Thesis work involved the porting, installation, modification and testing of a Hypercube simulation model using the SIMSCRIPT II.5 simulation language. By modifying the Supercube model, a 32 node Hypercube model was develop. The purpose of this testbed is to allow future students to experiment with different distributed query optimization heuristics which takes into account network traffic and network topology.

Schedules from four different query optimization algorithms were used to perform an investigation into the Hypercube Testbed. Over 160 experiments were conducted to determine:

- total execution time of a schedule.
- Response time of a schedule.
- effect relation location has on Total Time.
- effect relation location has on Response Time

Results of these experiments are submitted to [73] for publication.

This Thesis Report provides the background information needed to extend Hevner and Yao's algorithm to take into account network load and network topology.

Finally, this Thesis Report discusses the development of the testbeds, reports progress and documents many findings which resulted from verifying the DQDB and Hypercube simulation models.

6.2 FINDINGS DIRECTLY RELATED TO THE THESIS

Network descriptions, developed with the NETWORK II.5 simulation package, did not support the thesis.

- Owing to the fact that the DQDB MAC protocol logic was not provided as a standard bus access protocol in the NETWORK II.5 package, detailed DQDB protocol logic had to be developed for every node on the network.
- An excessive amount of code (i.e. 28,000 lines to define a 16 node network) had to be developed which ran slowly (i.e. approx. 15 hours on a 12MIP workstation to simulate a 1/4 second of real network time).
- The NETWORK II.5 user interface, which was designed to increase designer productivity became counter-productive with large network descriptions.
- Rigidity and lack of responsiveness of the NETWORK II.5 user interface made it necessary to develop alternative editing and debugging techniques.
- Other productivity tools such as Network Animation and On-line Plotting proved to be very limited in their usefulness.

Experimentation with the DQDB MAC MAN testbed did not support the thesis.

- The DQDB MAC MAN testbed proved to have a limited scope of applicability and limited flexibly. The software needed to simulate a distributed

database environment, to define characteristics of stored relations and queries, and to implement different query optimization routines would be difficult and time consuming to develop. These added features would increase the size of already huge programs while adding to their run length.

Programs developed with the SIMSCRIPT II.5 simulation language did support the thesis.

- SIMSCRIPT II.5 is a complete programming language with no other underlying languages. This provides the developer with more flexibility when developing simulation models. NETWORK II.5 uses SIMSCRIPT II.5 as an underlying language.
- SIMSCRIPT II.5 provides data structures such as: variables, arrays and sets. Access to these data structures allows complex models to be developed requiring less coding. These types of data structures are not available in NETWORK II.5.
- Compiled SIMSCRIPT II.5 programs run much faster than NETWORK II.5 descriptions. This allows larger, more complex models to be developed.
- When comparing SIMSCRIPT II.5 to NETWORK II.5, there is a trade-off between flexibility/efficiency and program development time. NETWORK II.5 allows programs to be developed in relatively short periods of time. However, the type of models which can be developed with NETWORK II.5 is very limited. These models also run very slowly. SIMSCRIPT on the

other hand, offers increased efficiency and flexibility at the cost of increasing development time.

Experimentation with the Hypercube testbed did support the thesis.

The Hypercube model shows (by construction) that it is possible to develop a testbed which will allow users to:

- Define a distributed database environment.
- Define the topology of the network on which the Distributed Database System is stored.
- Define traffic and other dynamic characteristics of the network at a give point in time.
- Experiment with different heuristics for query optimization.

6.3 OTHER FINDINGS

Experimentation with the DQDB MAC MAN showed it to be a very successful tool for studying MAN Performance.

Performance results from simulations that model a 4, 8 and 16 node network using Static Boundary, indicate that:

- the DQDB makes effective use of channel capacity.
- the DQDB exhibits better access delay characteristics than token ring.
- the DQDB exhibits better throughput characteristics than CSMA/CD.
- priority should be given to short control messages.

- nodes which are not located in close proximity to the frame generator have longer access delay characteristics without bandwidth balancing.

Performance results from the simulation that models a 16 node Broadcast network indicate that:

- bandwidth needs to be shared evenly at heavy loads.
- the Broadcast network utilizes 100 percent of the channel capacity but at only half the throughput.

Results from the Dynamic Boundary simulation show that:

- the Dynamic Boundary technique reduces both the average queue length and access delay.
- dynamic Boundary increases throughput.
- dynamic boundary offers a performance advantage over Static Boundary.
- these results are consistent with those obtained from queueing theory [64].

Results of simulating two simple bandwidth balancing techniques shows that:

- at light loads, Bandwidth Balancing techniques increase both access delay and queue length.
- at heavy loads, the Even Bandwidth balancing technique showed favorable results in sharing bandwidth over that of the Default Bandwidth Balancing technique.
- simulation results also show that Bandwidth Balancing techniques waste bandwidth.

The Distributed computing environment installed at the University of Windsor allowed large CPU intensive simulations to be run in parallel.

- By submitting jobs to different nodes in parallel work-load was distributed while also increasing job throughput.

Experimentation with the Hypercube testbed showed it to be a very successful tool for experimenting with different heuristics for query optimization.

Performance results from simulations that model different query optimization algorithms on a 32 node Hypercube network, indicate that:

- The SERIAL optimization algorithm produced a schedule that when simulated, was the least sensitive to network load and was the most successful in reducing total time for example 1.
- When the optimized SERIAL schedule for example 1 was executed, it also showed that this strategy performed significantly better when the network becomes overloaded.
- When simulating optimization Algorithms that do not take into account dynamic factors (i.e. network load and relation location) of the network, schedules maybe produced which perform worse than schedules that are not optimized (Example 2, Algorithm GENERAL TT).
- The parallel data transmissions in example 2 required considerable coordination in return for little reduction in schedule response time. This is consistent with the literature [31]

- Simple serial schedules performed as well or even better than more complicated parallel schedules. [31] states that simple distribution schedules are often beneficial because they require less coordination.
- Minimizing response time requires more data transmissions and leads to more queueing delays and synchronization delays. This is consistent with the literature [32].
- The JOIN strategy performed as well as the SEMI-JOIN strategy. The JOIN strategy requires less local processing than SEMI-JOIN. In high speed networks, operating at light to moderate load, local processing cost become dominant [71]
- The results indicate that in high speed networks, it might not be cost beneficial to use more complex optimization algorithms to reduce network cost further.

6.4 RECOMMENDATIONS

NETWORK II.5 Recommendations:

- Although NETWORK II.5 allows quick development of models and has a short learning cycle, NETWORK II.5 should only be used as a prototyping tool or as a performance evaluation tool for very small computer systems.
- NETWORK II.5 should only be used as a special purpose simulator of computer systems and Local Area Networks which have their protocols already programed within the package.

- the only LANs which should be simulated with this package are: 1) CSMA/CD, 2) CSMA/CA, 3) ALOHA, 4) Token Ring, 5) Slotted Token Ring (IEEE 802.4), 6) FDDI Priority Token Ring, and 7) IEEE 802.5 Priority Token Ring.
- NETWORK II.5 jobs should be submitted to execute under the UNIX ***NICE*** command (run a command at low priority).
- where ever possible, NETWORK II.5 jobs should be submitted in parallel to under utilized nodes in a distributed computing environment.

DQDB Protocol Recommendations:

- If the Bandwidth Balancing technique is implemented as a load sharing mechanism, it should only be activated when channel utilization approaches 100 percent.

SIMSCRIPT II.5 Recommendations:

- Although there is a trade-off between short learning cycles and program development times and software complexity and flexibility it is recommended that SIMSCRIPT II.5 be used for large complex models.

6.5 FUTURE WORK

The DQDB Testbed

Before beginning future work into developing new Bandwidth Balancing techniques, a detailed survey should be carried out to determine what work has been done in the area of Bandwidth Balancing on a Distributed Queue Dual Bus Network. Following this detailed

literature review, the DQDB testbed can be used to experiment with newly developed ideas and techniques.

Another fruitful area of research into the DQDB protocol is in the area of reuse of busy slots. Once a slot is marked as busy, it remains marked as busy for the duration of the time it is on the bus. It remains busy even after the receiving node has copied its contents from the bus. It is hoped that by reusing "stale" slots, throughput can be increased. Research into how this will affect the DQDB protocol logic (i.e. DQSM and various counters) and its effect on throughput is required.

The Hypercube Testbed

Before beginning to experiment with different heuristics for query optimization, a detailed survey should be carried out to determine what work has been done in the area of Distributed Query Optimization on High Speed Packet Switched Networks. Most of the work this author has been exposed to has been in the area of Distributed Query Optimization on LANs and on relatively slow WANs.

The query processing subsystem which implements the query optimization algorithm needs to be added to the model. Owing to time constraints, optimization algorithms were executed statically (outside the model). Resulting optimized schedules were hard code in the model so that testing of the testbed could be carried out.

A message passing module should be added to the model. It is stated in the literature that complex parallel strategies require more message passing. As a result of these messages, optimal strategies may actually be suboptimal. A message passing utility

should be added to the model to determine the effect of complex parallel schedule massage passing on execution times.

In the past calculating optimal distributed query execution strategies was based on intersite communication cost. The cost function to be minimized was restricted to the amount of data to be transmitted over the network. The transmission speed of all links were considered equal and individual link loading were not taken into account. The reason why these assumptions were made was that they simplified the optimized problem. A literature survey on this subject reveals that no papers exist which take into account network load when optimizing distributed queries. It seems reasonable that readily available network routing information can be used to optimize global queries.

As a starting point, new global query optimization algorithms should be developed which take network routing information into account. Algorithm GENERAL should be extended to take into account the load on individual network links. Next, studies should be undertaken to investigate and possibly prove that using network load information is cost beneficial when optimizing a query. As far as the author is aware, there exists no paper which states that this is not a fruitful area. If results of the investigation were not positive, then a published paper stating so would be extremely helpful.

Very little work has been done to determine if knowledge of network topology would be helpful in optimizing global queries. [74], [75] and [76] studied the broadcast capability of LANs to determine if it could be exploited to optimize join queries. [53] studies special algorithms which take into account the Star network topology and [77] studied query optimization algorithms for satellite networks. No work has been

documented which study packet switched public communication networks that determine the best site to carry out Semi-Join or Join operations. Work to determine site execution strategies and the exploitation of parallelism on these types of network topologies would be advantageous.

6.6 A VIEW TOWARDS THE FUTURE

This section briefly explores possible directions of Computer Networks and Distributed Systems.

An important component for transmitting data in modern communication networks and in those of the future is the optical fiber. Although there is already a base of optical fibers in place, research in solid-state physics in university and industrial laboratories is creating swift advances in this technology. Eric Sumner, vice president of AT&T Bell Laboratories™, has pointed out that fiber optic speeds have been doubling annually. If we assume a continuation of this doubling rate for the next decade of 1.7 Gbits, he indicates that we will be able to transmit all of human knowledge down a fiber in a few seconds. Areas that will require these high speeds are video and graphics communications. Some application of this technology will be High Definition TV (HDTV), CAD/CADD graphics, climate modeling, flight simulation and advanced manufacturing computing, to name just a few.

In the future, DDMSs will find universal acceptance because they address many of the following situations. First, most large organizations are geographically decentralized and have multiple computer systems at multiple locations. Second, in high transaction

rate environments one must assemble a large computing resource. While it is certainly acceptable to buy a large mainframe computer, it will be nearly 2 orders of magnitude cheaper to assemble a network of smaller machines and run a distributed system [4]. Also, owing to the fact that more users are expected to have workstations on their desk, DDMSs will be required to create a seamless connection between all of these users. Lastly, virtually all users must live with the sins of the past, i.e. data currently implemented in a multitude of previously generated systems. As a result, heterogeneous DDMSs will be required to tie these systems together.

In our increasingly global economy, corporations could use distributed computer systems to transact business more efficiently with one another and enjoy rapid, flexible and secure internal communication as well. New product designs and computer software, as well as economic, technological, recreational, financial and business information and services could be bought and sold in the information marketplace that would naturally evolve from advances in DDMSs and computer communication infrastructures.

We might even begin to collect the knowledge resources of our nation in a new kind of widely accessible electronic "Library of Congress." This leverage of increased and timely knowledge, if combined with faster and more flexible inter-organizational transactions toward the purchase and sale of goods and services, would undoubtedly enhance our industrial productivity across nearly all sectors. In this way we would be able to channel Western expertise and primacy in computers to help institutions, as well as product and service companies, across our entire economic front [4].

A vast amount of work in this area remains to be done if these visions are to be

realized.

REFERENCES

[1] C. Chung, "DATAPLEX: An access to heterogeneous distributed databases," *Comm. of the ACM*, vol. 33, no. 1, 1990.

[2] J. B. Rothnie and et al., "Introduction to a system for distributed databases SDD-1," *ACM Trans. on Database Systems*, vol. 5, no. 1, pp. 1–17, 1980.

[3] B. T. Hailpern and H. F. Korth, "An experimental database system for a network of personal workstations," in *Proceedings of the ACM Sigmod Database Week*, 1983.

[4] M. Stonebraker, "The design and implementation of distributed INGRES," in *The INGRES Papers: Anatomy of a Relational Database System*, pp. 187–196, Addison-Wesley Publishing Company, 1986.

[5] P. Gretton-Watson, "Distributed Database Development," *Comput. Comm.*, vol. 11, pp. 275–280, October 1988.

[6] T. Okoshi, "Recent Advances in Coherent Optical Fiber Communication Systems," *IEEE Jour. of Lightwave Technology*, vol. LT-5, January 1987.

[7] M. Schwartz, "Optical Fiber Transmission-From Conception to Prominence in 20 Years," *IEEE Comm. Magazine*, May 1984.

[8] M. J. F., "Metropolitan Area Network Update: The Global LAN is getting closer," *Data Communications Magazine*, pp. 111–123, 1989.

[9] H. F. Korth, *Database System Concepts*, pp. 401–479. McGraw-Hill, 1986.

[10] S. Ceri and G. Pelagatti, *Distributed Database Principles and Systems*. McGraw-Hill, 1984.

[11] S. Bandyopadhyay and A. Sengupta, "A Robust Protocol for Parallel Join Operation in Distributed Databases," in *Proceedings Int'l Symposium on Databases in Parallel and Distributed Systems*, pp. 97–106, IEEE Comput. Soc. Press, 1988.

[12] S. Bandyopadhyay, A. Sengupta, and A. Sen, "A Robust Protocol for Distributed Query Processing on Local Area Networks," tech. rep., Department of Computer Science, Windsor University, 1989.

[13] P. Maitra, "Performance Evaluation of Computer Networks Using Simulation," Master's thesis, Department of Computer Science, Arizona State University., December 1990.

[14] A. Sen, "Supercube: An Optimally Fault Tolerant Network Architecture," in *Acta Informatic 26*, pp. 741–748, 1989.

[15] N. F. Maxemchul, "Regular Mesh Topologies in Local and Metropolitan Area Networks," *AT&T Tech Jour.*, vol. 64, no. 7, pp. 1659–1684, 1985.

[16] A. S. Acampora, "A multichannel multihop local lightwave network," in *Proceedings of GLOBECOM 87*, pp. 51–51, 1987.

[17] CACI Products Company, La Jolla, California 92037, *NETWORK II.5 Users Manual*, 5th ed., August 1989.

[18] CACI Products Company, La Jolla, California 92037, *NETANIMATION Users Manual*, 1 ed., November 1987.

[19] CACI Products Company, La Jolla, California 92037, *SIMSCRIPT II.5 Programming Language*, 3rd ed., December 1988.

[20] CACI Products Company, La Jolla, California 92037, *SIMSCRIPT II.5 User's Guide and Casebook*, 2nd ed., August 1988.

[21] CACI Products Company, La Jolla, California 92037, *SIMSCRIPT II.5 Reference Handbook*, 3rd ed., January 1989.

[22] A. M. Law and C. S. Larmey, *An Introduction to Simulation Using SIMSCRIPT II.5*. Los Angeles, California: CACI Inc.- Federal, 1984.

[23] CACI Products Company, La Jolla, California 92037, *SIMSCRIPT II.5 UNIX User's Manual*, 1st ed., October 1989.

[24] E. C. Russell, *Building Simulation Models with SIMSCRIPT II.5*. Los Angeles, California: CACI Inc.- Federal, 1990.

[25] J. F. Mollenauer, "Networks for Greater Metropolitan Areas," *IEEE Communications Magazine*, pp. 115–128, February 1988.

[26] J. F. Mollenauer, "Standards for Metropolitan Area Networks," *IEEE Communications Magazine*, vol. 26, pp. 15–19, April 1988.

[27] R. M. Newman, Z. L. Budrikis, and J. L. Hullett, "The QPSX Man," *IEEE Communications Magazine*, vol. 26, pp. 15–19, April 1988.

[28] W. Jeon, C. Kim, and K. Kim, "Design of a Distributed Isochronous Channel Management Protocol," in *Infocom. 89 IEEE CH2702-9*, pp. 268–275, 1989.

[29] J. L. Hullett and P. Evans, "New Proposal Extends the Reach of Metro Area Nets," *Data Communications*, pp. 139–147, February 1988.

[30] S. G. Popovich, M. Alam, and S. Bandyopadhyay, "Simulation of a Distributed Queue Dual Bus (IEEE 802.6) Protocol using NETWORK II.5," in *Proceedings, Twenty-Second Annual Pittsburgh Conference on Modeling and Simulation*, (Pittsburgh PA.), May 1991.

[31] A. R. Hevner and S. B. Yao, "Query Processing in Distributed Database Systems," *IEEE Transactions on Software Engineering*, vol. SE-5, pp. 177–187, May 1979.

[32] P. M. G. Apers, A. R. Hevner, and S. B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Transactions on Software Engineering*, vol. SE-9, pp. 57–68, January 1983.

[33] M. T. Ozsu, "Distributed Database Systems - A Reason for Chaos or a Vehicle for Order," in *Proceedings Chaos into Order. CIPS Edmonton*, pp. 111–120, Canadian Inf. Process. Soc, 1989.

[34] E. Wong, "Dynamic Rematerialization: Processing Distributed Queries Using Redundant Data," *IEEE Transactions on Software Engineering*, vol. SE-9, pp. 228–232, May 1983.

[35] D. L. Eager and K. C. Sevcik, "Achieving Robustness in Distributed Database Systems," *ACM Transactions on Database Systems*, vol. 8, pp. 354–381, September 1983.

[36] S. B. Davidson, H. Garcie-Molina, and D. Skeen, "Consistency in Partitioned Networks," *Computing Surveys*, vol. 17, September 1985.

[37] B. T. Blaustein and C. W. Kaufman, "Updating Replicated Data During Communications Failures," in *Proceedings Eleventh Int. Conf. Very Large Databases*, pp. 49–58, Aug 1985.

[38] J. D. Ullman, *Database and Knowledge-base Systems*, vol. 1, pp. 543–600. Computer Science Press, 1988.

[39] P. Bodorik, "Distributed Query Processing Optimization Objectives," in *Proceedings Fourth International Conference on Data Engineering (CAT. No. 88CH2280-2)*, (Los Angeles, CA, USA), pp. 320–329, Washington, DC, USA: IEEE Comput. Soc. Press, February 1988.

[40] C. W. Chung and K. B. Irani, "An Optimization of Queries in Distributed Database Systems," *Journal of Parallel and Distrib. Comput.*, vol. 3, pp. 137–157, June 1986.

[41] L. S. Colby, "A Recursive Algebra and Query Optimization for Nested Relations," *ACM Computing Surveys*, vol. 16, no. 2, pp. 273–283, 1989.

[42] W. W. Chu, "Optimal File Allocation in a Multiple Computer System," *IEEE-TC*, vol. C-18, no. 10, 1969.

[43] R. G. Casey, "Allocation of copies of a File in an Information Network," *Proceedings AFIPS Spring Nat. Comput. Conf.*, vol. 40, pp. 617–625, 1972.

[44] C. V. Ramamoorthy and B. W. Wah, "The Isomorphism of Simple File Allocation," *IEEE Transactions on Computers*, vol. C-32, March 1983.

[45] A. E. Abbadi, D. Skeen, and F. Cristian, "An Efficient, Fault-Tolerant Protocol for Replicated Data Management," *ACM*, 1985.

[46] E. F. Codd, "A Relational Model of Data for Shared Data Banks," *Comm. ACM*, vol. 13, pp. 377–387, 1970.

[47] X. Wang and W. S. Luk, "Parallel Join Algorithms on a Network of Workstations," in *Proceedings Int'l Symposium on Databases in Parallel and Distributed Systems*, pp. 87–95, IEEE Comput. Soc. Press, 1988.

[48] N. Goodman, P. A. Bernstein, E. Wong, C. L. Reeve, and J. B. Rothnie, "Query processing in SDD-I: A system for distributed databases," *ACM-TODS*, vol. 6, no. 4, 1981.

[49] K. B. Irani and A. Knabbaz, "A Methodology for the Designing of Communication Networks and the Distribution of Data in Distributed Supercomputer Systems," *IEEE-TC*, vol. C-31, no. 5, 1982.

[50] C. T. Yu and C. C. Chang, "Distributed Query Processing," *Computing Survey*, vol. 16, no. 4, pp. 399–433, 1984.

[51] R. Epstein, M. Stonebraker, and E. Wong, "Distributed Query Processing in a Relational Database System," in *Proceedings, International Conference on Management of Data*, ACM-Sigmod, May 1978.

[52] R. Epstein and M. Stonebraker, "Analysis of Distributed Database Processing Strategies," in *Proceedings of the International Conference on Very Large Databases*, pp. 92–110, 1980.

[53] L. Kershberg, P. D. Ting, and S. B. Yao, "Query Optimization in Star Computer Networks," *ACM Transactions on Database Systems*, vol. 7, pp. 678–711, December 1982.

[54] P. M. G. Apers, "Critique on and Improvement of Hevner and Yao's Distributed Query Processing Algorithm G," internal report, Vrije University, Amsterdam, The Netherlands, 1979.

[55] W. W. Chu and P. Hurley, "Optimal Query Processing for Distributed Database Systems," *IEEE Transactions on Computers*, vol. C-30, pp. 835–850, September 1982.

[56] W. Stallings, *Data and Computer Communications*, vol. 2. New York: Macmillan, 1988.

[57] A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[58] R. E. Ross, "FDDI - a Tutorial," *IEEE Communications Magazine*, vol. 24, pp. 10–23, May 1986.

[59] W. E. Burr, "FDDI Optical Data Link," *IEEE Communications Magazine*, vol. 24, pp. 8–23, May 1986.

[60] A. S. Acampora and M. Karol, "An overview of lightwave packet networks," *IEEE Network Magazine*, pp. 29–41, January 1989.

[61] P. A. Morreale, "Metropolitan-Area Networks," *IEEE Spectrum*, pp. 40–42, May 1990.

[62] M. Sloman and J. Kramer, *Distributed Systems and Computer Networks*. Prentice-Hall Int'l, 1987.

[63] S. G. Popovich, "Simulation of DQDB Metropolitan Area Networks using NETWORK II.5," tech. rep., Department of Computer Science, University of Windsor, Windsor, Ontario, Canada, September 1990.

[64] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*. Reading, MA: Addison-Wesley, 1988.

[65] K. Kummerle, "Multiplexer Performance for Integrated Line and Packet-Switched Traffic," in *ICCC, Stockholm*, pp. 508–515, 1974.

[66] IEEE 802.6 Standards Committee, *Proposed Standard: Distributed Queue Dual Bus (DQDB) Metropolitan Area Network (MAN)*, 1989. Unapproved Draft - Published for Comment Only.

[67] C. J. Weinstein, M. L. Malpass, and M. J. Fischer, "Data Traffic Performance of an Integrated Circuit and Packet-Switched Multiplex Structure," *IEEE Trans. on Comm.*, vol. 6, June 1980.

[68] J. Squire and S. M. Palais, "Programming and Design Considerations of a Highly Parallel Computer," in *AFIPS Spring Joint Computer Conference*, pp. 395–400, 1963.

[69] D. Bertsekas and R. Gallager, *Data Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[70] P. Valduriez and G. Gardarin, "Join and Semi-Join Algorithms for a Multi Processor Database Machine," *ACM Trans. Database Systems*, vol. 9, March 1984.

[71] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[72] S. G. Popovich, S. Bandyopadhyay, and M. Alam, "Simulation of a DQDB MAC Protocol with Movable Boundary and Bandwidth Balancing Mechanisms." Submitted to the Simulation Multi Conference for publication, April 4-6, 1992, November 1991.

[73] S. G. Popovich, S. Bandyopadhyay, and M. Alam, "Simulation of SEMI-JOIN and JOIN Optimization Schedules on a Packet Switch Hypercube Network." Submitted to the Twenty-Third Annual Pittsburgh Conference on Modeling and Simulation, May, 1992, November 1991.

[74] Z. Ozsoyoglu and N. Zhou, "Distributed Query Processing in Broadcast Local Area Networks," in *Proceedings 20th Hawaii International Concerence on System Sciences*, (Hailua-Kona, Hawaii), pp. 419–429, 1987.

[75] T. Page and G. Popek, "Distributed Data Management in Local Area Networks," in *Proceedings of ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, (Portland, OR), pp. 135–142, 1985.

[76] B. W. Wah and Y. N. Lien, "Design of Distributed Databases on Local Computer Systems with a Multiaccess Network," *IEEE Trans. Softw. Eng*, vol. SE-11, pp. 606–619, July 1985.

[77] J. Lachimia, "Query Decomposition Distributed Database System Using Satellite Communications," in *Proceedings of the 3rd Seminar on Distributed Data Sharing Systems*, (Parma Italy), pp. 105–118, 1984.

APPENDIX A

GLOSSARY OF NETWORK TERMS

Access Control Field[10]

The protocol control information in a slot, which is used to support the access control function.

Bandwidth

The difference between the limiting frequencies of a continuous frequency spectrum.

Bandwidth balancing mechanism

A procedure to facilitate effective sharing of the bandwidth, whereby a node occasionally bypasses the use of empty Queued Arbitrated Slots.

Baseband cable

A transmission medium that is used for digital transmission. The bandwidth possible depends on the cable length. For 1 Km cables, a data rate of 10 Mbits/s is possible. Higher data rates are possible on shorter cables. Coaxial cables are widely used for local area networks.

[10] The terms in this glossary are taken from the Glossary of Telecommunication Terms, FED-STD-1037A, National Communications Systems, 1986, [56], [57] and [64].

Broadband cable

Is a transmission medium that is used for analog transmission. It is also referred to as broadband. Since broadband networks use analog signaling cables can run nearly 100 Km. Broadband offers multiple channels, which are capable of transmitting 3 Mbits/s each.

Broadcast

The simultaneous transmission of data to a number of stations.

Broadcast Communication Network

A communication network in which a transmission from one station is broadcast to and received by all other stations.

Bus

One or more conductors that serve as a common connection for a related group of devices.

Busy slot

A slot which contains information and is not available for Queued Arbitrated Access.

CCITT

Abbreviation for "International Telegraph and Telephone Consultative Committee."

Circuit switching

A method of communicating in which a dedicated communications path is established between two devices through one or more intermediate switching nodes. Unlike packet switching, digital data are sent as a continuous stream of bits. Bandwidth is guaranteed,

and delay is essentially limited to propagation time. The telephone system uses circuit switching.

Communications architecture

The hardware and software structure that implements the communications function.

Communication network

A collection of interconnected functional units that provides a data communications service among stations attached to the network.

Connectivity

This is defined as the minimum number of nodes whose removal from the node-set would leave the graph disconnected or trivial.

CSMA

Carrier Sense Multiple Access. A medium access control technique for multiple-access transmission media. A station wishing to transmit first senses the medium and transmits only if the medium is idle.

CSMA/CD

Carrier Sense Multiple Access with Collision Detection. A refinement of CSMA in which a station ceases transmission if it detects a collision.

Datagram service

Service at the Network Layer in which successive packets may be routed independently from end to end. There is no call setup phase. Datagrams may arrive out of order.

Data link Layer

Layer 2 of the OSI model. Converts an unreliable transmission channel into a reliable one.

Degree

The degree of a node, is the number of edges (links) connected to it.

Delay time

The sum of waiting time and service time in a queue.

Diameter

This is the maximum distance between two nodes.

Distance

This is the length of the shortest path between two nodes.

Frame

Unit of data of the Data link Layer.

Frequency-division multiplexing

The division of a transmission facility into two or more channels by splitting the frequency band transmitted by the facility into narrower bands, each of which is used to constitute a distinct channel.

Hamming distance

The number of bit positions in which two code words differ is called the hamming distance. Given the code words 00001 and 11000, the hamming distance is 3.

High-speed local network (HSLN)

Header

System-defined control information that precedes user data.

High-speed local network (HSLN)

A local network designed to provide high throughput between expensive, high-speed devices, such as mainframes and mass storage devices.

Integrated services digital network

A planned worldwide telecommunication service that will use digital transmission and switching technology to support voice and digital data communication. ISDN is an abbreviation for "Integrated Services Digital Networks." All- digital network handling a multiplicity of services with standard interfaces for user access.

Isochronous

The time characteristic of an event or signal recurring at known, periodic time intervals.

Local area network (LAN)

A general-purpose local network that can serve a variety of devices. Typically used for terminals, microcomputers, and mini-computers within a small area.

Medium access control (MAC)

For broadcast networks, the method of determining which device has access to the transmission medium at any time. CSMA/CD and token are common access methods.

Multiplexing

In data transmission, a function that permits two or more data sources to share a common transmission medium such that each data source has its own channel.

Network Layer

Layer 3 of the OSI model. Responsible for routing data through a communications network.

Optical fiber

A thin filament of glass or other transparent material through which a signal-encoded light beam may be transmitted by means of total internal reflection.

OSI

Abbreviation for "Open System Interconnection."

Packets

Small blocks of data into which messages are broken, for transmission across a network.

Packet switching

A method of transmitting messages through a communication network, in which long messages are subdivided into short packets. The packets are then transmitted as in message switching. Usually, packet switching is more efficient and rapid than message switching.

Physical Layer

Layer 1 of the OSI model. Concerned with the electrical, mechanical, and timing aspects of signal transmission over a medium.

Point-to-point

A configuration in which two stations share a transmission path.

Poisson process

One of the most common arrival processes in queueing theory which has the memoryless property of successive arrivals.

Presentation Layer

Layer 6 of the OSI model. Concerned with data format and display.

Protocol

A set of rules that govern the operation of functional units to achieve communication.

Public data communication network

A government-controlled or national monopoly packet-switched network. This service is publicly available to data processing users.

Segment

The protocol data unit of 52 bytes transferred between peer DQDB layer entities as the information payload of a slot. It contains a segment header of 4 bytes and a segment payload of 48 bytes. There are two types of segments: Pre-Arbitrated segments and Queued Arbitrated segments.

Session Layer

Layer 5 of the OSI model. Manages a logical connection (session) between two communicating processes or applications.

Shortest path

A least cost path between a given source-destination pair.

Sliding-window technique

A method of flow control in which a transmitting station may send numbered packets within a window of numbers. The window changes dynamically to allow additional packets to be sent.

Switched communication network

A communication network consisting of a network of nodes connected by point-to-point links. Data are transmitted from source to destination through intermediate nodes.

Time-division multiplexing (TDM)

The division of a transmission facility into two or more channels by allotting the facility to several different information channels, one at a time.

Token bus

A medium access control technique for bus/tree. Stations form a logical ring, around which a token is passed. A station receiving the token may transmit data, and then must pass the token on to the next station in the ring.

Token ring

A medium access control technique for rings. A token circulates around the ring. A station may transmit by seizing the token, inserting a packet onto the ring, and then retransmitting the token.

Topology

The structure, consisting of paths and switches, that provides the communications interconnection among nodes of a network.

Transmission medium

The physical path between transmitters and receivers in a communications system.

Transport Layer

Layer 4 of the OSI model. Provides reliable, transparent transfer of data between endpoints.

Twisted pair

A transmission medium consisting of two insulated wires arranged in a regular spiral pattern.

Virtual circuit

A packet-switching service in which a connection (virtual circuit) is established between two stations at the start of transmission. All packets follow the same route, need not carry a complete address, and arrive in sequence.

APPENDIX B

AN OVERVIEW OF THE NETWORK II.5 SIMULATION PACKAGE

B.1 INTRODUCTION

Simulation is an effective way of pretesting proposed systems, plans, or policies before developing expensive prototypes, field tests, or actual implementations. Using computer-based simulation, it is possible to trace out in detail the consequences and implications of a proposed course of action.

Simulation is often used as an alternative to more traditional forms of analysis such as analytical solutions (e.g. closed-form solutions to steady-state queueing models), numerical solutions (e.g. to differential equations), or even to scale-model building (e.g. for harbors or waterway systems).

The Goal of NETWORK II.5

The goal of the NETWORK II.5 simulation tool is to provide those people who are developing or refining computer and/or communication networks with a tool that will let them study the operation of their proposed system without having to study a simulation language.

NETWORK II.5 is a design tool which takes a user-specified computer system description and provides measures of hardware utilization, software execution, and conflicts.

The NETWORK II.5 software package consists of three main parts: a front-end input file processor (NETIN); the program that actually runs the simulation (NETWORK); and third, an interpreter of the results of the simulation (NETPLOT). The computer network to be simulated is described by data structures consisting of Processing.Elements (nodes or computers), Transfer.Devices (buses, fiber optic cabling, etc.), modules (software instructions) and files. Each of these building blocks has a series of attributes whose values will be set to model the proposed network.

All of the functions related to building and maintaining simulation-data-files (description of the hardware and software of the system to be simulated) are performed by the NETIN program. NETWORK is the program that reads the simulation-data-file, builds and then executes the simulation. To monitor system activity as the simulation progresses, fourteen different reports are provided and can be displayed on any standard terminal or printer. The NETPLOT program provides a time line status report (post-processed) showing, for each module, message, and hardware device selected, the times at which the device was busy. Output may cover the entire simulation period, or smaller periods which provide greater detail can be examined.

B.2 SIMULATION COMPONENTS

Hardware Components

Hardware devices are specified in NETWORK II.5 by using building blocks based on the functions of the devices being modelled. There are three functions performed by the hardware elements in a computer system. These are:

- *Process data* — performed by a Processing.Element building block
- *Transfer data* — performed by a Transfer.Device building block
- *Store data* — performed by a Storage.Device building block

These building blocks are powerful enough to model any device that performs the given function, since the building blocks model a function by its timing. For example, to model a communication link, we need only to know when to send the data and how long it takes to move the data from source to destination. Whether the technology involved is a fiber optic link or a satellite channel is immaterial.

A Processing.Element (PE) is used to model a data source/sink, a bus controller, a display, a sensor which generates interrupts, or an entire arithmetic logic unit. A PE is characterized by its Instruction.Repertoire, Message.Queue.Size, I/O.Setup.Time, Basic.Cycle.Time, Time.Slice, Interrupt.Overhead and Input.Controller.

Instructions define what a Processing.Element can do. Instructions form a link between the hardware (PEs) and software (modules). The correspondence is established through user defined names. When a module is executed on a particular PE, it issues the instructions it has in its Instruction.List by name to the PE, in the order that they

appear. The PE is then responsible for examining its Instruction.Repertoire to find an instruction definition by that name. The instruction definition contains all of the information describing the instruction's type and characteristics.

Transfer.Devices are links connecting Processing.Elements and Storage.Devices. They are used to move data between Processing.Elements or between a Processing.Element and a Storage.Device. They can connect as many of these devices as desired. Each Transfer.Device has a user-defined specification giving the transfer speed, transfer overhead and protocol definition.

Data is moved between Processing.Elements over a Transfer.Device as a result of a Message.Instruction. A user has to specify the amount of time required to transfer a word and transfer a block. This allows a user to model up to two levels of a data packet structure by means of a user specified word overhead time and block overhead time. The user-defined block overhead time can be used for adding destination headers to message packets, adding message checksums, etc.

There are many pre-programmed protocols which dictate how a Transfer.Device will actually carry out the transfer of data. These are:

- FCFS protocol (First Come, First Served)
- Collision Protocol
- Priority Protocol
- Token Ring Protocol
- Slotted Token Ring Protocol
- Priority Token Ring Protocol

- Aloha Protocol

Storage.Devices contain both user-named files and unstructured storage (called General.Storage). They have a capacity measured in bits. For simulations where the file structure has yet to be determined, files can be read from or written to a Storage.Device's General.Storage. General.Storage keeps track of the number of bits stored, but not the individual file names.

Many applications require choosing a value at run time from a user specified statistical distribution. Statistical.Distributions are referenced by their user-defined name. NETWORK II.5 supports the following types of distributions:

- Exponential Gamma
- IEEE Backoff
- Key Linear
- Log Normal
- Normal
- Erlang
- Uniform

Software Components

Messages are commonly used to schedule and coordinate tasks. A message is sent from one PE to another over a Transfer.Device as a result of a Message.Instruction. When a message is received, it is always filed in the receiving PE's Received.Message.List. Once in that list, modules with that message in their Required.Message.List will contend

for that message. The highest priority module that has all of its preconditions met will be assigned the message. In the event of a tie, the module that waited the longest will get the message.

A message can be viewed as a token. One message token can satisfy no more than one module's Required.Message precondition. Multiple copies of the same message token may reside in the same Received.Message list until taken by a module.

A message is defined implicitly through the definition of Message.Instructions. In the instruction, a message is described by the Message.Text, and the length in bits. The length is used in determining message transmission times over the various Transfer.Devices.

Semaphores are also used to schedule and coordinate tasks. Every semaphore has a count associated with it. Semaphores can be incremented, decremented or assigned a semaphore's count by using a semaphore instruction. Scheduling a task by the use of a semaphore differs from scheduling by messages in that:

1. Scheduling is instantaneous (no transfer time).
2. All modules monitoring the semaphore will immediately take action (messages start only one module).
3. Semaphore actions are timed.

The software components of a system are characterized in NETWORK II.5 as Modules, Instruction.Mixes, Macro.Instructions and Files.

A module is the specification of a task to be performed by a Processing.Element. The module description consist of four parts:

1. Scheduling conditions
2. Host Processing.Elements options
3. A list of instructions to execute
4. A list of module(s) to execute when this module completes

Each of these parts maps into a stage that every module goes through in its "life". The stages in a module life are:

1. Checking preconditions - every module must meet certain conditions which enable it to run. These conditions are often based on time and system state.
2. Requesting a host PE - to execute a module must acquire a host PE.
3. Executing instructions - a module executes by sequentially issuing all of the instructions in its instruction list.
4. Choosing successors - after all instructions in a module have executed, the module is marked as completed and any successor modules are automatically activated.

B.3 REPORTING

NETWORK II.5 offers a user fourteen different reports. The reports fall into four basic categories:

- Playback
- Runtime-interactive
- Summary

- Post-interactive

Playback reports are solely the result of an analysis of a user's input data file. They are produced before the simulation commences and are requested during the interactive dialogue session in NETIN.

The runtime interactive reports are requested by a user when setting up a simulation run or by setting the proper global flags for batch runs. These reports follow the simulation as it progresses. They allow a user to monitor the progress of the system simulated to ensure that it is progressing properly. By monitoring the simulation, it can be immediately stopped if the simulation is in trouble.

Summary reports provide totals, averages, maximums, minimums, standard deviations, etc., for the simulation run from the start of the simulation to the time of the report. They are produced by two different mechanisms. During the NETWORK II.5 interactive simulation setup dialogue, a set of summary reports can be defined to be produced periodically during a simulation run. In addition, a complete set of summary reports is always produced at the end of the simulation. In all cases these reports are routed to the list file. The summary reports include the following:

- Processing.Elements Utilization Statistics
- Instruction Execution
- Transfer.Device Utilization
- Storage.Device Utilization
- Complete Module Summary
- Semaphore Report

- Message Statistics
- Received Message Report

In order to give an overview of the utilization of the various hardware devices and semaphore values as a function of time, a post processor with graphical output, called NETPLOT, has been included in the NETWORK II.5 package. NETPLOT uses a data file written during a NETWORK II.5 simulation run to produce timeline status plots and utilization plots on a file or terminal. The plot range is specified by the user, and the plot may be routed to the user's terminal, a file, or both.

B.4 ANIMATION

NetAnimation brings animated display to NETWORK II.5 simulations, making it possible to actually see the modeled computer system operate. It is easy to define the location, color and style of every device in a NETWORK II.5 simulation. The animation may be advanced a single step at a time or can proceed automatically; and trace messages describing simulation activity may be displayed concurrent with the simulation. NetAnimation uses the standard plot and listing files generated by all implementations of NETWORK II.5 as input, so a simulation executed on a mainframe can be animated on a personal computer.

APPENDIX C

AN OVERVIEW OF THE SIMSCRIPT II.5 SIMULATION LANGUAGE

C.1 DISCRETE-EVENT SIMULATION

Discrete-event simulation describes a system in terms of logical relationships that cause changes of state at discrete points in time rather than continuously over time. Examples of problems in this area are most queueing situations: Objects (customers in a gas station, aircraft on a runway, jobs in a computer) arrive and change the state of the system instantaneously. State variables are such things as the number of objects waiting for service and the number being served. One could argue, of course, that these changes also appear to occur continuously.

Things that can go wrong with a simulation are:

1. Failure to Define an Achievable Goal
2. Incomplete Mix of Essential Skills
3. Inadequate Level of User Participation
4. Inappropriate Level of Detail
5. Poor Communication
6. Using the Wrong Computer Language
7. Obsolete or Nonexistent Documentation
8. Using an Unverified Model
9. Failure to Use Modern Tools and Techniques to Manage the Development of a Large Complex Computer Program
10. Using Mysterious Results

The Model Structure

Simulation models exhibit many common properties. Every model has the following three ingredients:

1. A mechanism for representing arrivals of new objects.
2. The representation of what happens to the objects within the modelled system.
3. A mechanism for terminating the simulation.

The Process Concept

A process represents an object and the sequence of actions it experiences throughout its life in the model. There may be many instances (or copies) of a process in a simulation.

There may also be many different processes in a model.

A process object enters a model at an explicit simulated time, its "creation time." It becomes active either immediately or at a prescribed "activation time." From then on, the description of its activity is contained in the process routine. A process routine may be thought of as a sequence of interrelated events separated by lapses of time, either predetermined or indefinite.

Predetermined lapses of time are used to model such phenomena as the service time (deterministic or stochastic), whereas indefinite delays arise because of competition between processes for limited resources. In this latter a case process will automatically be delayed until the resource is made available to it.

The Resource Concept

Resources are the passive elements of a model. A resource is used to model an object which is required by the process objects. If the resource is not available when required, the process object is placed in a queue or waiting line and made to wait until the resource becomes available.

C.2 PROGRAM STRUCTURE

A SIMSCRIPT II.5 program consists of three primary elements:

a. A preamble giving a static description of each modelling element

b. A main program where execution begins.

c. A process routine for each process declared in the preamble.

Variables

SIMSCRIPT variables are of numerous types. Attributes, reference variables, and simple variables are only a few of the types available, but they will suffice to introduce the concepts.

Variables are either global or local in scope. A global variable must be defined in the program preamble. If a variable is not explicitly defined anywhere, it becomes a local variable by default.

Random Number Generation

The heart of any representation of random phenomena on a computer is a source of random numbers. In SIMSCRIPT II.5 a function called RANDOM.F serves this purpose. It is a generator of numbers which are uniformly distributed on the open interval (0, 1). RANDOM.F is referred to as a source of pseudo-random numbers, since any sequence of numbers can be reproduced at will merely by initializing the generator function to the same starting seed. The technique upon which RANDOM.F is based is the Lehmer technique. In this method, a starting seed is multiplied by a constant to produce a new seed and a sample. The constant is chosen as a function of the size of the computer word and thus is different for different architectures. Therefore, the same model using random numbers may yield slightly different results on different computers. The starting seed is actually a variable which has a default non-zero value.

External Processes

Sometimes data are available about an explicit sequence of process activations to be

included in a simulation. Sources for such data might be:

1. Observations of an actual system
2. Tapes generated from a computer performance measurement system
3. Historical files such as weather records

Analogy Between Processes and Events

An event, in SIMSCRIPT, is described by the same triple as a process:

1. An event "notice" – the data block containing pertinent information about a particular copy of (or instance of) the event.
2. An event routine – the program describing the logic of the event, how it responds to its environment, and how it changes the environment.
3. A global variable with the same name as the event – used to identify a particular instance of the event, either at creation time or during execution.

A process may be thought of as a sequence of events. To replace even the most elementary process by the equivalent event code, it will require at least two events to represent a simulated time delay.

APPENDIX D

LIST OF PUBLICATIONS RESULTING FROM THIS THESIS WORK

[1] S. G. Popovich, M. Alam, and S. Bandyopadhyay, "Simulation of a Distributed Queue Dual Bus (IEEE 802.6) Protocol using NETWORK II.5," in *Proceedings, Twenty-Second Annual Pittsburgh Conference on Modeling and Simulation*, (Pittsburgh PA), May 1991.

[2] S. G. Popovich, M. Alam, and S. Bandyopadhyay, "Simulation of a DQDB MAC Protocol with Movable Boundary and Bandwidth Balancing Mechanisms," in *Proceedings, Simulation Multi Conference*, (Orlando FL), April 6–9, 1992.

[3] S. G. Popovich, S. Bandyopadhyay, and M. Alam, "Simulation of SEMI-JOIN and JOIN Query Optimization Schedules on a Packet Switched Hypercube Network," in *Proceedings, Twenty-Third Annual Pittsburgh Conference on Modeling and Simulation*, (Pittsburgh PA), May 1992.

APPENDIX E

M. Sc. THESIS DEFENCE SLIDE PRESENTATION

M.Sc. THESIS DEFENCE

BY
STEVEN POPOVICH

Thesis Title:

"A Testbed for Distributed Query Optimization"

COMMITTEE:

Dr. S. Bandyopadhyay - Supervisor
Dr. R Frost/Dr. M. Alam - Dept. Reader
Dr. H. Kwan - External Reader

PURPOSE AND OBJECTIVES

- PURPOSE OF THIS PRESENTATION:
  - is to present the project work and results of the investigation for the purpose of supporting my thesis statement.
- MY OBJECTIVES ARE TO:
  - state my thesis
  - support the thesis statement by:
    - presenting the work needed to perform the investigation.
    - present results of the investigation.
    - summarize the results that support the thesis statement.

AGENDA

- STATE THE THESIS STATEMENT (What is it?)
- BACKGROUND AND OVERVIEW
  - what is a distributed system?
  - what is a DDMS + components?
  - why this project is important.
- DESCRIPTION OF THE PROJECT
  - objectives + scope for the project.
  - completed objectives (what was done?).
- REVIEW THE DQDB INVESTIGATION (1st TESTBED)
  - review work performed.
  - summarize important results.
- THE HYPERCUBE INVESTIGATION (2nd TESTBED)
  - the hypercube model
  - experimenting with distributed queries
  - results + findings
- SUMMARY OF RESULTS THAT SUPPORT THE THESIS (What was found?)

THE THESIS STATEMENT

- it is possible to develop a simulation testbed which will allow users to:
  - define a distributed database environment.
  - define the topology of the network on which the DDS is stored.
  - define traffic + other dynamic characteristics of the network at a given point in time.
  - define the characteristics of stored relations and queries.
  - experiment with different heuristics for query optimization.
- PURPOSE OF THE PROJECT
  - The purpose of this investigation is to develop a testbed, written with the aid of simulation software, which will enable users to experiment with different distributed query optimization heuristics.

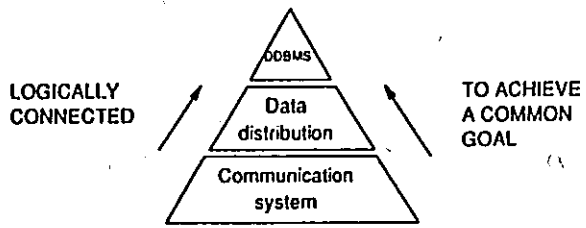
WHAT IS A DISTRIBUTED SYSTEM?

DEFINITION:

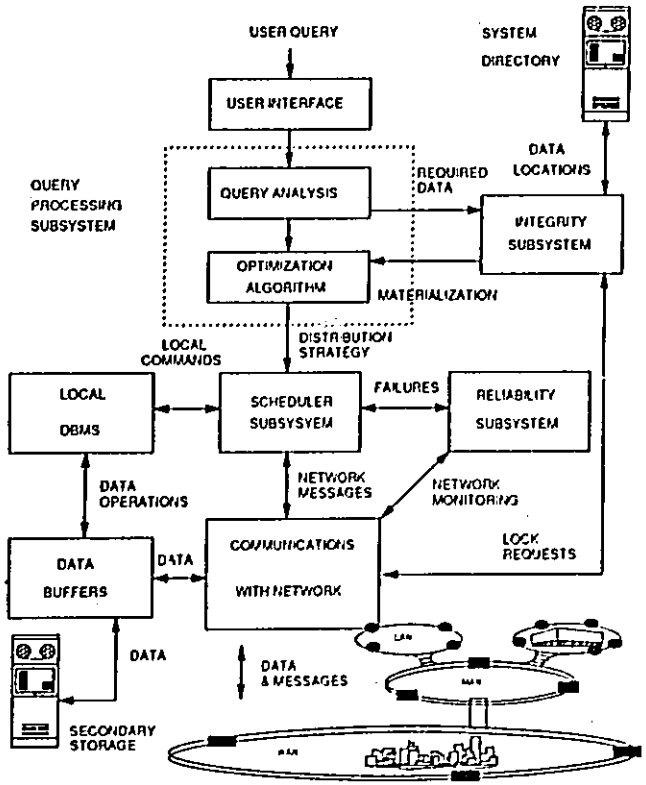
A Distributed System is one in which several autonomous processors and data stores supporting processes and/or databases cooperate to achieve an overall goal. Processes coordinate their activities and exchange information over a communications network.

ATTRIBUTES:

- The communication network allows for the interconnection of shared resources
- Distribution of data and processors
- DDBMS - software that integrates the distributed elements into a "single" coherent system



DDBMS - MAJOR SUBSYSTEMS

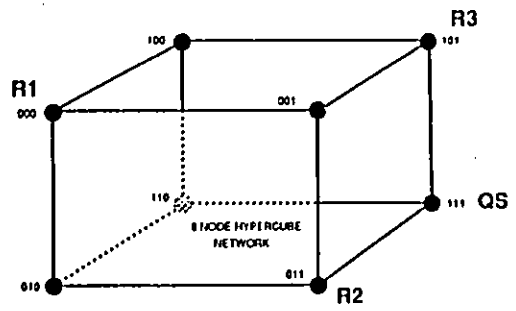


WHY IS THIS TYPE OF INVESTIGATION IMPORTANT?

- **IF DDBMSs ARE TO GAIN WIDE ACCEPTANCE QUERIES MUST EXECUTE IN A TIMELY FASHION**
  - query response time must be minimized (user waiting time).
  - distributed queries must execute efficiently (reduce load on the network).
  - parallelism should be exploited whenever possible.
- **CURRENTLY THE NETWORK ROUTING MANAGER AND THE DDBMS ARE TWO SEPERATE NON-INTERACTING PIECES OF SOFTWARE**
  - this was acceptable in the past because of relatively few network architectures were available.
  - amount of data being put online is increasing.
  - user sophistication increasing (banking).

WHY IS THIS TYPE OF INVESTIGATION IMPORTANT?

- **COMPLEX COMMUNICATION ARCHITECTURES ARE NOW TECHNOLOGICALLY FEASIBLE**
  - data rates doubling every year (fiber).
  - architectures now allow more than one path between a source and destination.
  - can query optimizers use the knowledge of network architecture and traffic conditions to optimize queries?
- **WE NEED A FLEXIBLE TESTBED THAT WILL ENABLE USERS TO TEST/INVESTIGATE NEW IDEAS (INSIGHT)**
  - what is a reasonable heuristic?
  - is the benefit worth the cost?



M.Sc. THESIS DEFENCE

WHAT WAS DONE?

START

ATTEND A.S.U. STUDY
SIMULATION +
NETWORK ARCHITECTURE

- NETWORK II.5 ON SUN/4
- DEVELP. DQDB MAN.
- DEVELP. DETAILED LOGIC
- PERFORM INVESTIGATION

- DEVELP. 8 DQDB ARCHTCTRS.
- 19 DIFFERENT PROGRAMS
- PERFORMED INVESTIGATION
- OVER 150 EXPERIMENTS

- SIMSCRIPT II.5 INSTALLED ON SUN/4

- PORT + MODEFIED SUPERCUBE MODEL
- DEVELP. 32 NODE HYPERCUBE

- ADD QUERY SCHEDULING SYS. TO THE HYPERCUBE MODEL.
- GENERATE OPT. SCHEDULES ALGO SER. PARA. GEN. JOIN.

- USE RESULTING SCHEDULES TO TEST THE TESTBED
- OVER 100 EXPERIMENTS

- THESIS REPORT
- THESIS DEFENCE

- EXPLORE SIMSCRIPT II.5 SIMULATION LANGUAGE FOR THE PURPOSE OF CREATING THE SECOND TESTBED
- EXPLORE THE SUPERCUBE MODEL FOR THE PURPOSE OF CREATING THE TESTBED
- PORT + MODIFY THE SUPERCUBE VAX/UNIX MODEL TO THE SUN4/UNIX ENVIRONMENT
- DEVELOP A TESTBED WHICH MODELS A HYPERCUBE NETWORK WITH 32 NODES.
- ADD A QUERY SCHEDULING SUBSYSTEM TO THE MODEL
- USE RESULTING OPTIMIZED DISTRIBUTION SCHEDULES FROM ALGORITHM SERIAL, PARALLEL GENERAL AND JOIN TO TEST THE THE HYPERCUBE TESTBED

- EXPLORE THE NETWORK II.5 SIMULATION PACKAGE FOR THE PURPOSE OF CREATING A TESTBED
- DEVELOP A TESTBED THAT MODELS A DISTRIBUTED QUEUE DUAL BUS MAN
  - develop + document revised DQDB protocol logic
  - simulate a standard DQDB MAN (4, 8, 16 nodes + 3 programs)
- SIMULATE A BROADCAST DQDB (WITH SB + DBWB)
- SIMULATE A STANDARD DQDB (WITH SB + UBWB)
- SIMULATE A STANDARD DQDB (WITH SB + EBWB)
- SIMULATE A STANDARD DQDB (WITH DB + DBWB)
- SIMULATE A STANDARD DQDB (WITH DB + EBWB)
- PERFORM AN INVESTIGATION INTO DIFFERENT DQDB ARCHITECTURES TO DETERMINE:
  - avg. packet queueing time + avg. slot queueing time
  - avg. queue length + throughput
  - network capacity + max operating range
  - utilization + queueing time by location on network

ONE DAY

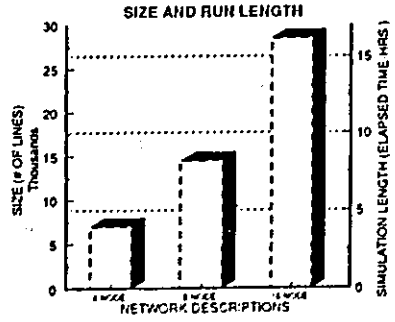
SGP 1091

SL # 12

M.Sc. THESIS DEFENCE

FINDINGS DIRECTLY RELATED TO THE THESIS

- NETWORK DESCRIPTIONS, DEVELOPED WITH THE NETWORK II.5 PACKAGE DID NOT SUPPORT THE THESIS
  - DQDB protocol logic not provided as a standard bus access protocol.
  - an excessive amount of code had to be developed which ran slowly.
  - NETWORK II.5 interface became cumbersome productive with large descriptions.
  - alternative editing and debugging ha[illegible]e used.
- EXPERIMENTING WITH THE DQDB MAC MAN TESTBED DID NOT SUPPORT THE THESIS
  - limited scope of applicability/limited flexibility
  - query optimization algorithms/scheds. would be very time consuming to develop.
  - would increase the size of already huge programs while adding to the run length.



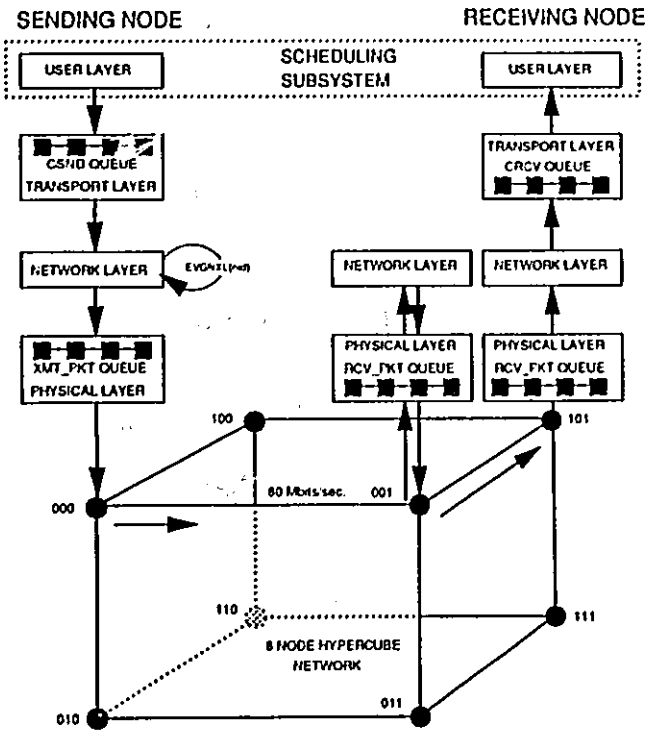
SGP 1091

SL # 11

OTHER FINDING (SUMMARY)

- THE INVESTIGATION INTO THE DQDB MAC MAN SHOWS:
  - the DQDB exhibits better access delay characteristics than token ring.
  - the DQDB exhibits better throughput characteristics the CSMA/CD.
  - priority should be given to short messages.
  - bandwidth not shared evenly.
  - the DB technique reduces both average queue length and access delay.
  - DB increases throughput [64].
  - DB offers a performance advantage over SB.
  - these results are consistent with those obtained from queueing theory [64].
  - at light loads, BWB techniques increase both access delay and queue length.
  - simulation results show that BWB techniques waste bandwidth.

THE HYPERCUBE MODEL



M.Sc. THESIS DEFENCE

DISTRIBUTION SCHEDULES FOR THE IFS, ALGORITHMS PARALLEL AND SERIAL

MATERIALIZATION

Relations listed in ascending order of size (S_i)

Relation R_i	Size S_i	Selectivity P_i
R1: ON ORDER	200	0.2
R2: SUPPLIER	400	0.4
R3: PARTS	600	0.6
R4: COST	1000	1.0

Setup cost = 20.

Location of relations in network

NODE 6 - R1 ON ORDER

NODE 7 - R2 SUPPLIER

32 NODE HYPERCUBE NETWORK

NODE 15 - R3 PARTS

NODE 23 - R4 COST

NODE 31 - QUERY SITE

Schedule for the Intial Feasible Solution (IFS)

R1: 220 / (200 + 20) E

R2: 420 / (400 + 20) E

R3: 620 / (600 + 20) E

R4: 1020 / (1000 + 20) E

Theoretical Response time = 1020

Theoretical Total Time = 2280

Schedule for Algorithm PARALLEL (min. response time)

R1: 220 / (200 + 20) E — Schedule selected as minimum

R2: 420 / (400 + 20) E

R1: 220, R3: 100 / (2 * 400) + 20 E — Schedule selected as minimum

R3: 620 / (600 + 20) E

R1: 220, R3: 140 / (2 * 600) + 20 E — Schedule selected as minimum

R1: 220, R2: 100, R3: 68 / (2 * 400) + 20, (2 * .4 * 600) + 20 E

All R4 schedules not shown — Schedule selected as minimum

R1: 220, R2: 100, R4: 100 / (2 * 400) + 20, (2 * .4 * 1000) + 20 E

Schedule for Algorithm PARALLEL (min. response time)

R1: 220, R3: 140 / (2 * 600) + 20 E

R1: 220, R2: 100, R4: 100 / (2 * 400) + 20, (2 * .4 * 1000) + 20 E

Theoretical Response time = 423 (60%)

Theoretical Total Time = 780 (65%)

Schedule for Algorithm SERIAL (min. total time)

R1: 220, R2: 100, R3: 68, R4: 68 / (2 * 400) + 20, (2 * .4 * 600) + 20, (2 * .4 * 1000) + 20

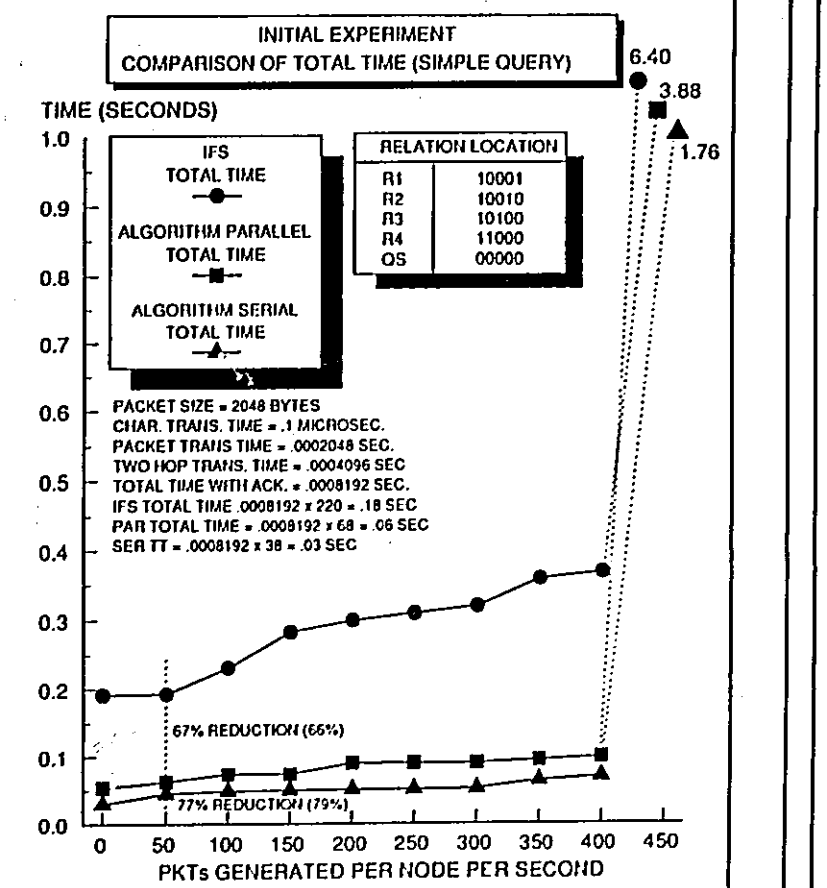
Theoretical Response time = 458 (55%)

Theoretical Total Time = 458 (80%)

SGP 10/91

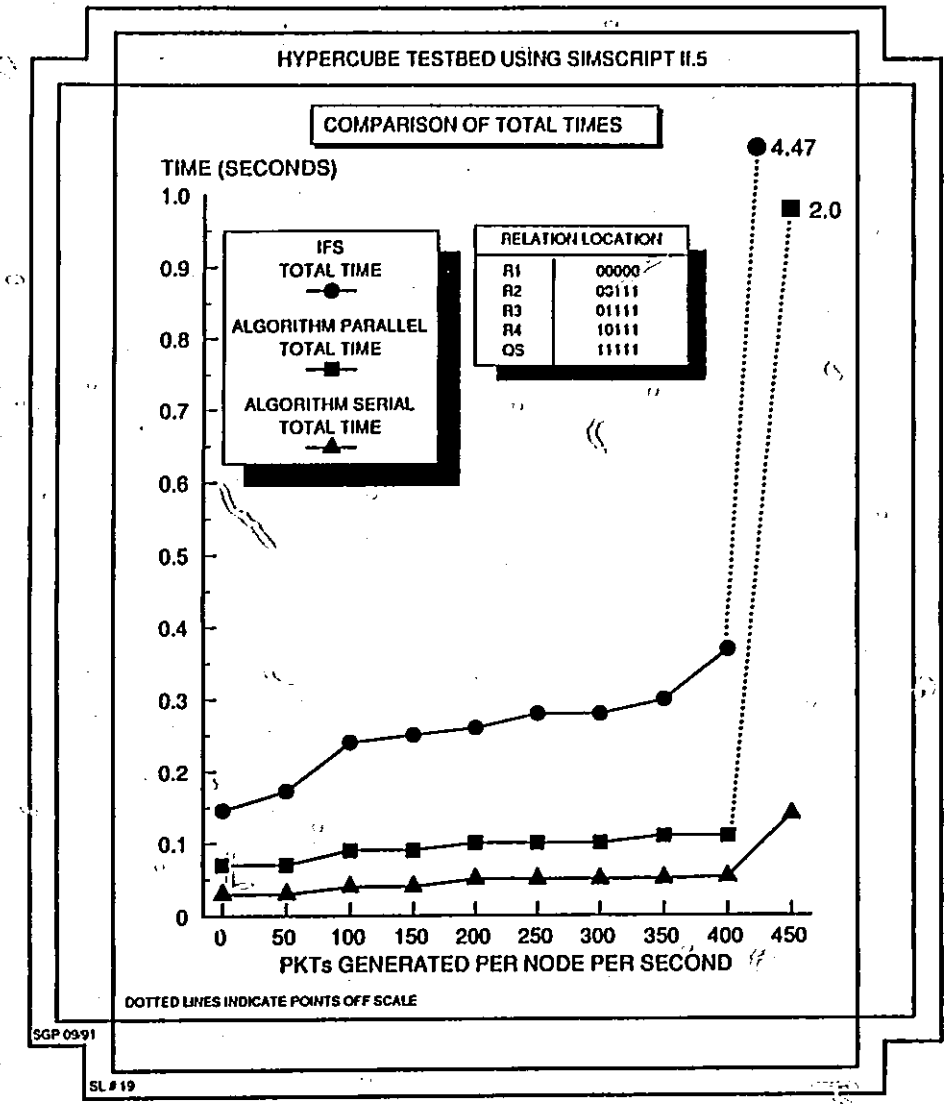
SL #

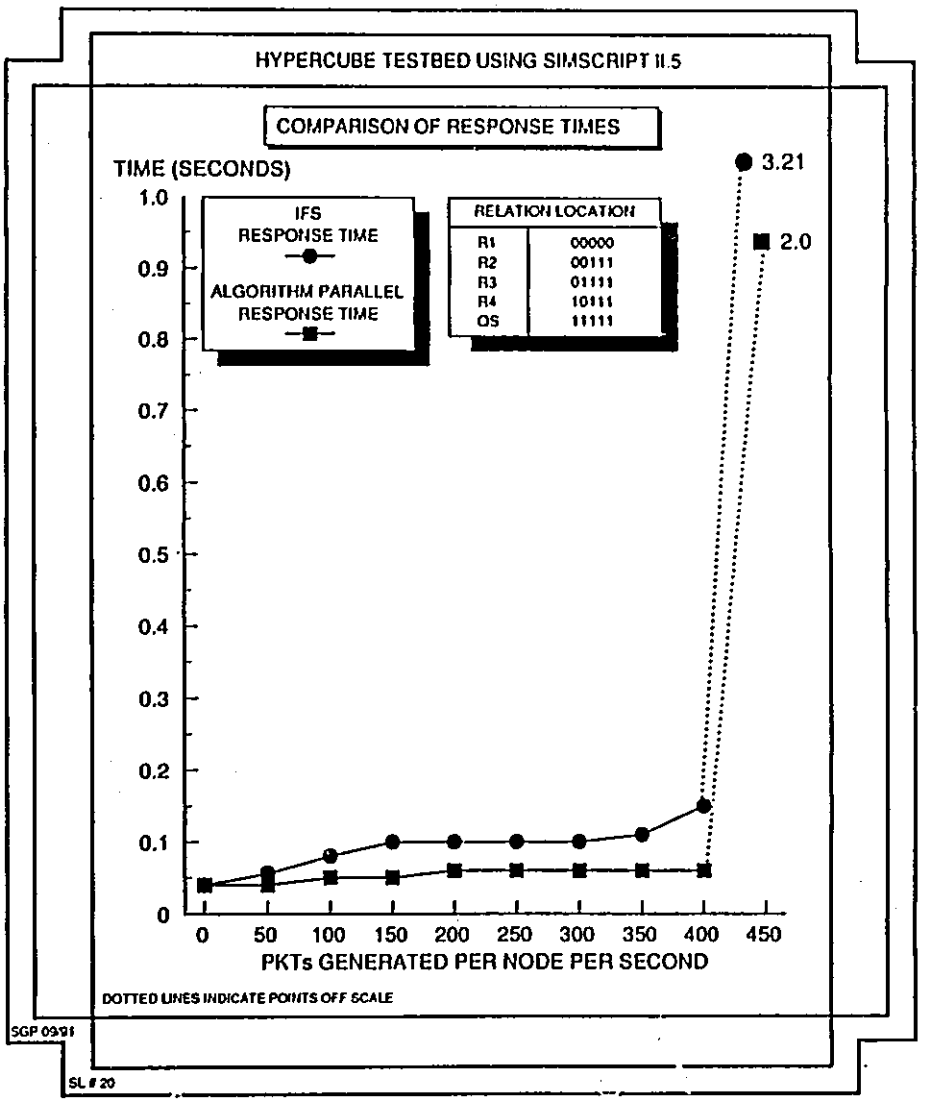
M.Sc. THESIS DEFENCE



SGP 10/91

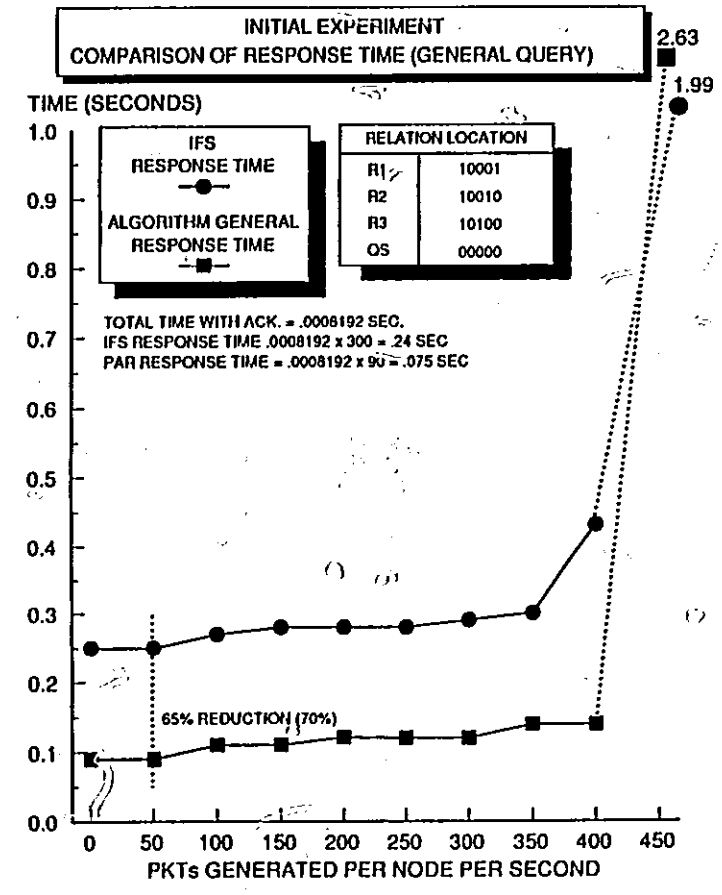
SL #


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF TOTAL TIMES
TIME (SECONDS)
4.47
2.0
IFS
TOTAL TIME
ALGORITHM PARALLEL
TOTAL TIME
ALGORITHM SERIAL
TOTAL TIME
RELATION LOCATION
R1 00000
R2 00111
R3 01111
R4 10111
QS 11111
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 09/91
SL # 19


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF RESPONSE TIMES
TIME (SECONDS)
3.21
2.0
IFS
RESPONSE TIME
ALGORITHM PARALLEL
RESPONSE TIME
RELATION LOCATION
R1 00000
R2 00111
R3 01111
R4 10111
QS 11111
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 09/91
SL # 20

M.Sc. THESIS DEFENCE

INITIAL EXPERIMENT
COMPARISON OF RESPONSE TIME (GENERAL QUERY)



SGP 10-91

SL # 18

177

M.Sc. THESIS DEFENCE

DISTRIBUTION SCHEDULES FOR THE IFS, ALGORITHM GENERAL AND THE JOIN STRATEGY

MATERIALIZATION

Relations listed in ascending order of size (S_i)

Relation R_i	Size S_i	Join Attribute d_{i1} = P# b_{i1}	P_{i1}	d_{i2} = S# b_{i2}	P_{i2}
R1: ON-ORDER	1000	400	0.4	100	0.2
R2: SUPPLIER	2000	400	0.4	450	0.9
R3: PARTS	3000	900	0.9	-	-

Setup cost = 20

Location of relations in network

NODE 0 - R1 ON-ORDER (S#,P#,QTY)

NODE 15 - P2 SUPPLIER (S#,P#,Jn)

32 NODE HYPERCUBE NETWORK

NODE 23 - R3 PARTS (P#,PNAME)

NODE 31 - QUERY SITE (P#,PNAME,SUM(QTY))

Schedule for the Inital Feasible Solution (IFS)

R1 1020 — 1000 + 20 — E

R2 2020 — 2000 + 20 — E

R3 3020 — 3000 + 20 — E

Theoretical Response time = 3020

Theoretical Total Time = 6060

Schedule for Algorithm GENERAL (min response time)

R2 R1

R1 d_{12} 120 d_{22} 110 — 380 — E

R2 d_{21} 420 — (9 * 4 * 1000) + 20

R1 d_{12} 120 — R2 — 420 — E

(2 * 2000) + 20

R1 d_{11} 420 — R3 — 500 — E

R2 d_{21} 420 — (4 * 4 * 3000) + 20

Theoretical Response time = 920 (70%)

Theoretical Total Time = 2910 (52%)

Schedule for Join Strategy (min total time)

R1 1020 — R2 180 — R3 100 — E

(4 * 2 * 2000) + 20

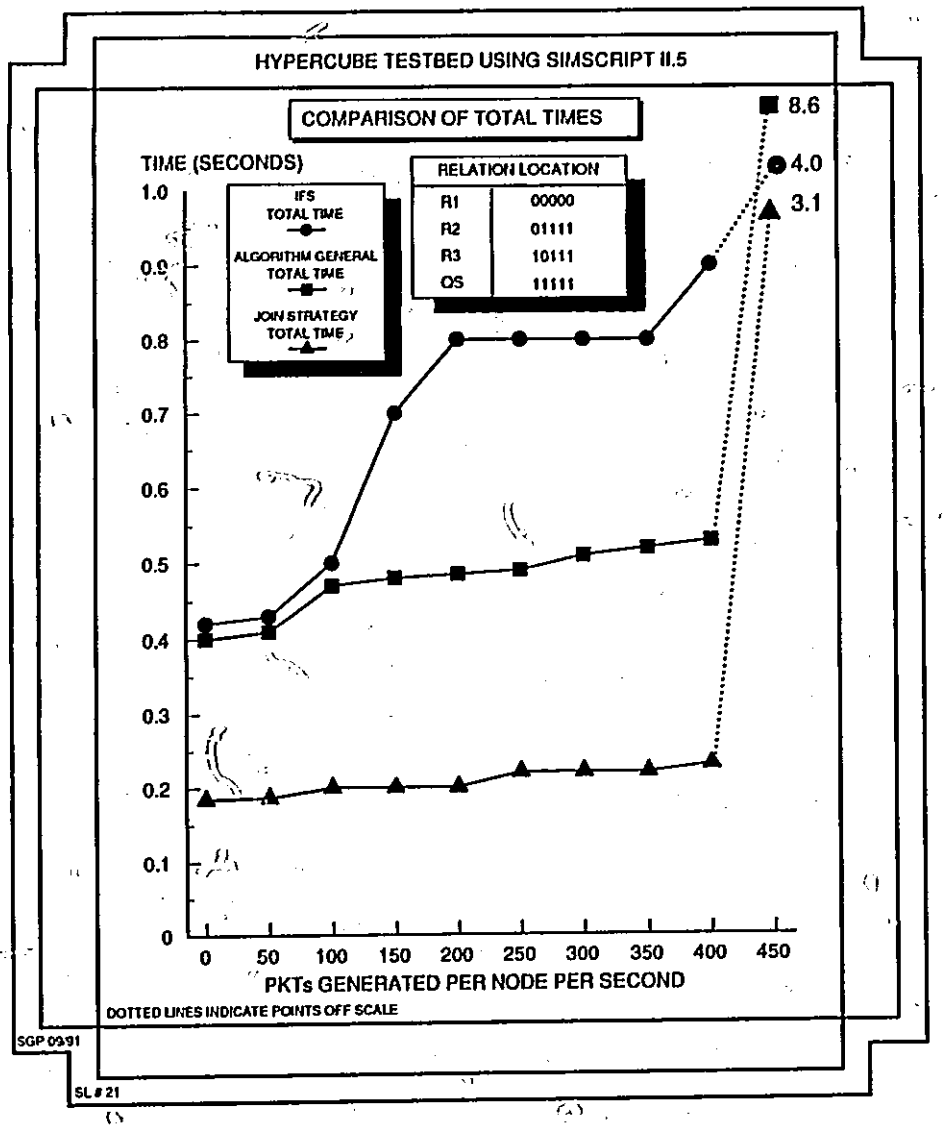
(4 * 2 * 4 * 3000) + 20

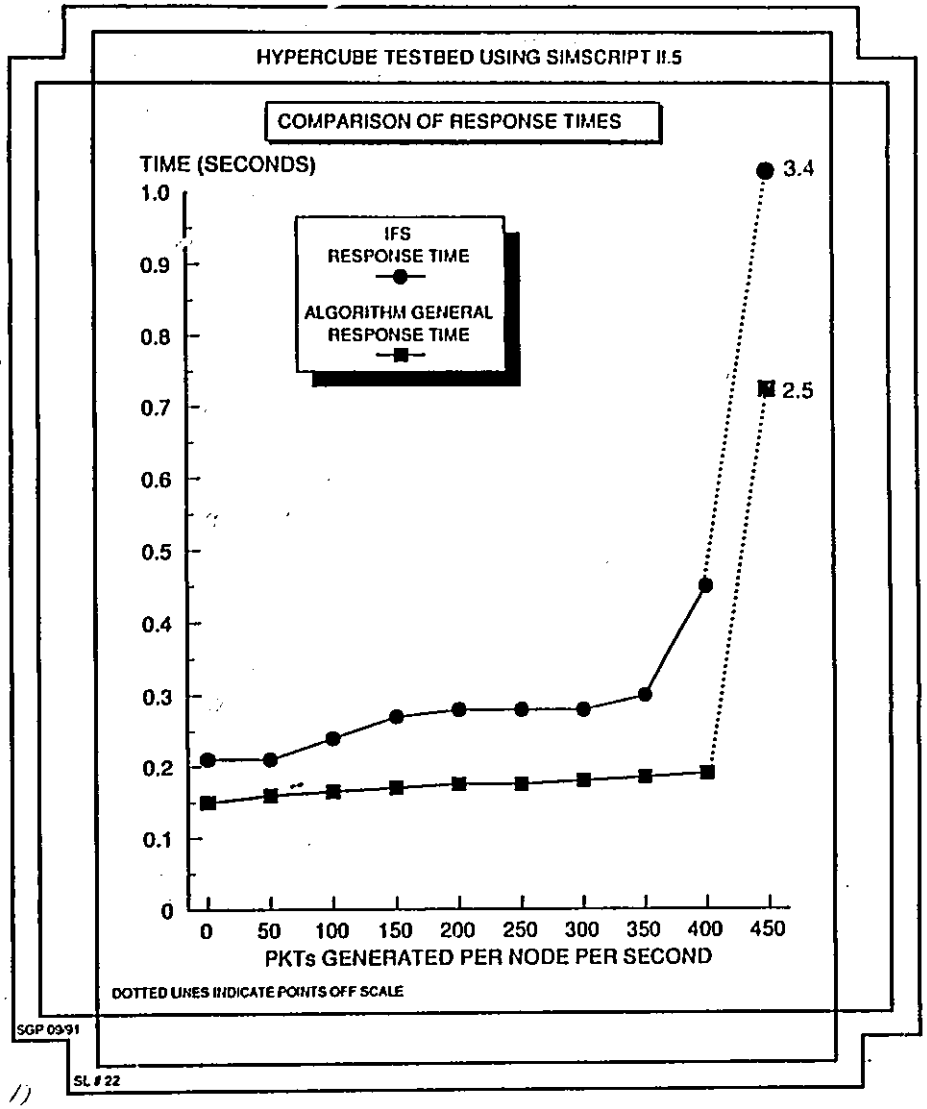
Theoretical Response time = 1306 (55%)

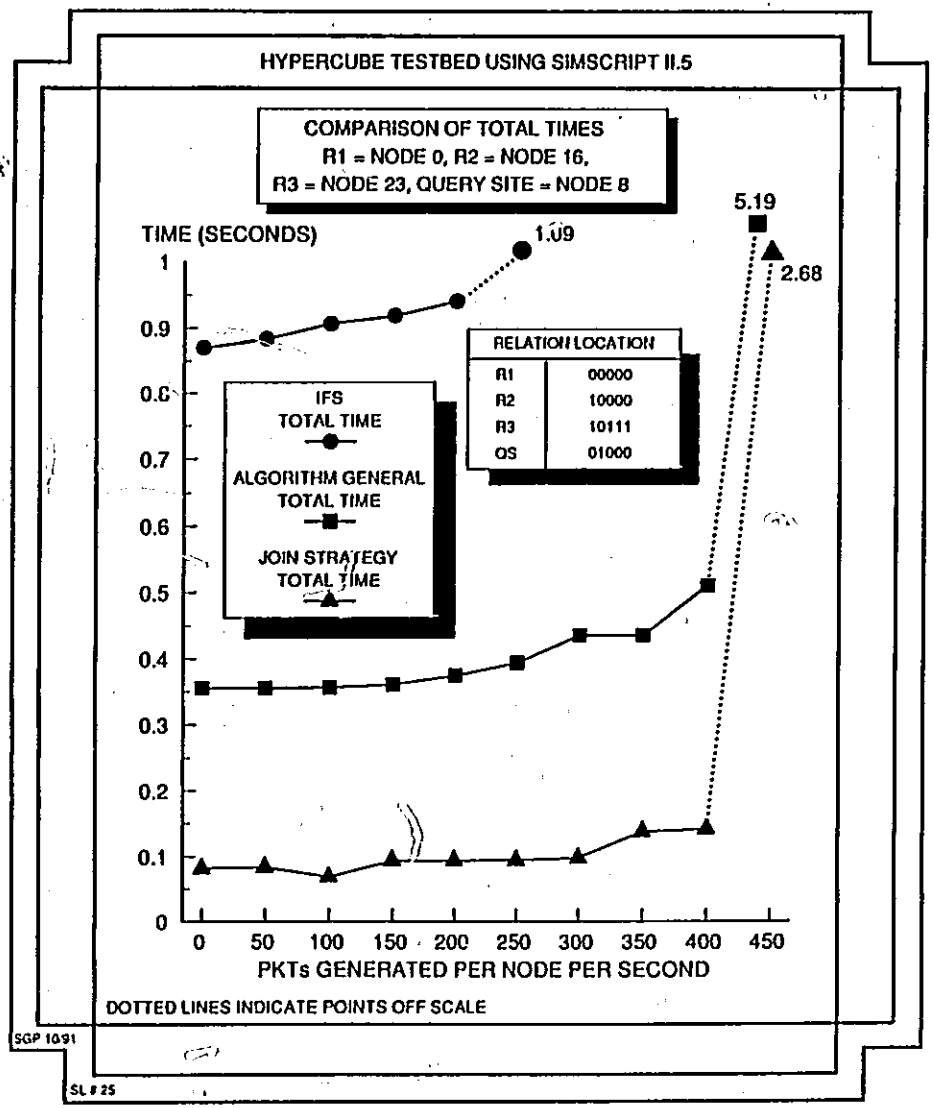
Theoretical Total Time = 1306 (80%)

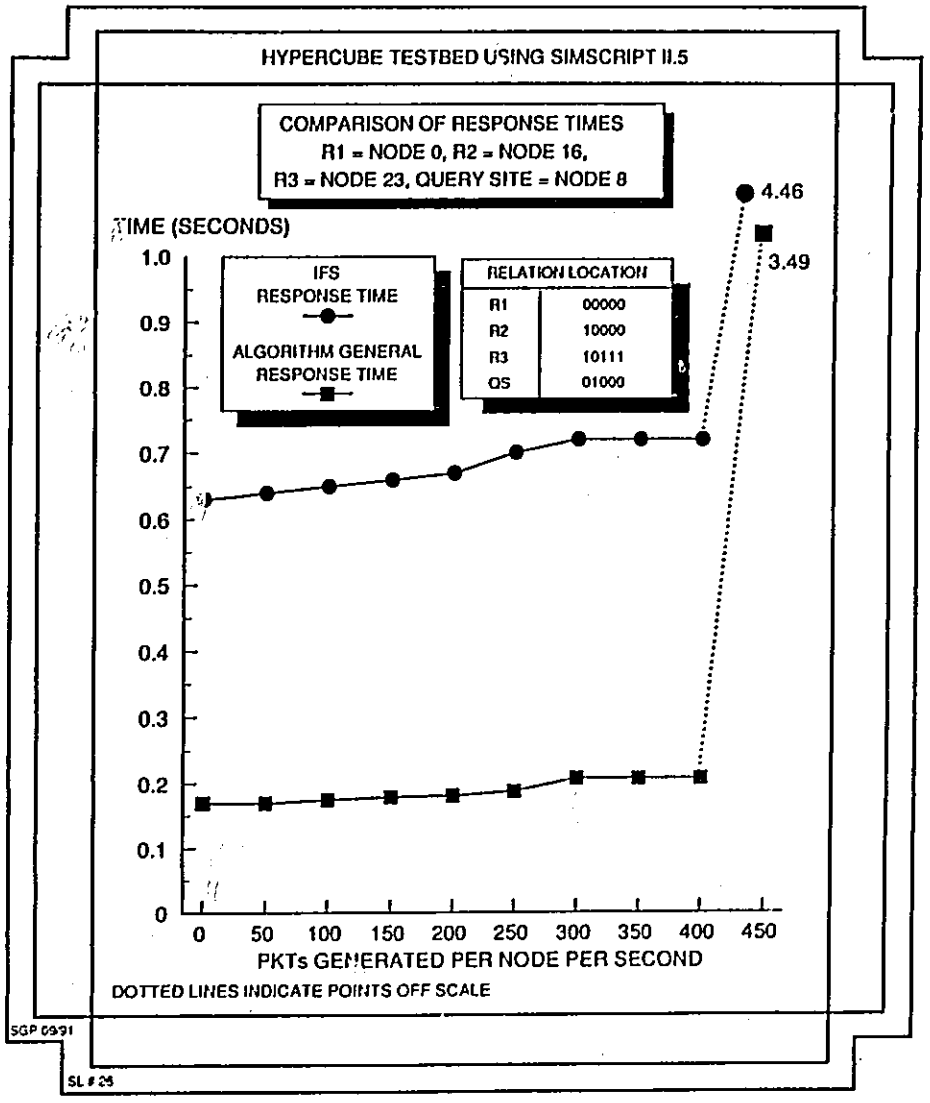
SGP 10-91

SL #


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF TOTAL TIMES
TIME (SECONDS)
IFS TOTAL TIME
ALGORITHM GENERAL TOTAL TIME
JOIN STRATEGY TOTAL TIME
RELATION LOCATION
R1 00000
R2 01111
R3 10111
OS 11111
8.6
4.0
3.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 03/91
SL # 21


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF RESPONSE TIMES
TIME (SECONDS)
IFS RESPONSE TIME
ALGORITHM GENERAL RESPONSE TIME
3.4
2.5
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 03/91
SL # 22


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF TOTAL TIMES
R1 = NODE 0, R2 = NODE 16,
R3 = NODE 23, QUERY SITE = NODE 8
TIME (SECONDS)
1.09
5.19
2.68
IFS
TOTAL TIME
ALGORITHM GENERAL
TOTAL TIME
JOIN STRATEGY
TOTAL TIME
RELATION LOCATION
R1 00000
R2 10000
R3 10111
QS 01000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 10/91
SL # 25


HYPERCUBE TESTBED USING SIMSCRIPT II.5
COMPARISON OF RESPONSE TIMES
R1 = NODE 0, R2 = NODE 16,
R3 = NODE 23, QUERY SITE = NODE 8
TIME (SECONDS)
4.46
3.49
IFS
RESPONSE TIME
ALGORITHM GENERAL
RESPONSE TIME
RELATION LOCATION
R1 00000
R2 10000
R3 10111
QS 01000
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 50 100 150 200 250 300 350 400 450
PKTs GENERATED PER NODE PER SECOND
DOTTED LINES INDICATE POINTS OFF SCALE
SGP 09/91
SL # 25

FINDINGS DIRECTLY RELATED TO THE THESIS

- PROGRAMS DEVELOPED WITH THE SIMSCRIPT II.5 SIMULATION LANGUAGE, DID SUPPORT THE THESIS
  - more flexibility in developing simulation models.
  - access to data structures (variables, arrays, sets) allows complex models to be develop requiring less coding.
  - compiled SIMSCRIPT II.5 programs run much faster than NETWORK II.5 descriptions.
  - tradeoff between flexibility/efficiency and program development time.
- EXPERIMENTING WITH THE HYPERCUBE TESTBED DID SUPPORT THE TESTBED
  - the results show that it is possible to develop a testbed which will allow users to:
    - define a distributed database environment
    - define the topology of the network on which the DDBS is stored.
    - define traffic and other dynamic characteristics of the network at a given point in time.
    - experiment with different heuristics for query optimization.
- EXPERIMENTATION WITH THE HYPERCUBE TESTBED SHOWED IT TO BE A VERY SUCCESSFUL TOOL FOR EXPERIMENTING WITH DIFFERENT HEURISTICS FOR QUERY OPTIMIZATION

CONTRIBUTIONS OF THIS THESIS WORK

- THE FIRST CONTRIBUTION OF THIS THESIS WORK INVOLVED THE DEVELOPMENT OF A DQDB MAN SIMULATION MODEL USING THE NETWORK II.5 SIMULATION PACKAGE.
  - The testbed is available to future students to continue the investigation into the DQDB MAC MAN protocol.
    - bandwidth balancing
    - reuse of slots
- THE SECOND CONTRIBUTION OF THIS THESIS WORK INVOLVED THE DEVELOPMENT OF A HYPERCUBE SIMULATION MODEL USING SIMSCRIPT II.5 SIMULATION LANGUAGE FOR THE PURPOSE OF EXPERIMENTING WITH DISTRIBUTED QUERY OPTIMIZATION HEURISTICS
  - The second testbed supports the thesis.
  - The second testbed is more flexible.
  - Executes faster.
  - More amenable to changes (user friendly?)

APPENDIX F

SOURCE CODE FOR THE SIMULATION PROGRAMS (SEPARATELY BOUND)

The source listing of the DQDB and Hypercube simulation programs are bound separately and can be obtained from the School of Computer Science at the University of Windsor.

APPENDIX G

GENERAL BIBLIOGRAPHY (SEPARATELY BOUND)

A complete bibliography is bound separately and can be obtained from the School of Computer Science at the University of Windsor.

APPENDIX H

VITA AUCTORIS

Steven Popovich was born in Scranton, Pennsylvania. He graduated from McDonald Cartier High School — Montreal, Quebec in 1978. From there Steve joined Dawson College where he obtained a Diploma of Collegial Studies in Mechanical Engineering Technology in 1983. Following this, Steve graduated with a Certificate in Quality Control in 1985 and a Bachelor's Degree in Computer Science in 1987 from Concordia University — Montreal, Quebec. Upon graduation, he joined Dow Chemical Canada Inc. as an Information Systems Analyst in Sarnia, Ontario. He is currently a candidate for a Master's degree in Computer Science at the University of Windsor and will complete all degree requirements in the Fall of 1991. Steve intends to return to Dow Chemical Canada Inc. as a Senior Technology Analyst and work in the networking and distributed systems area. He is a member of both the IEEE and American Society for Quality Control.