Rudowsky, Ira Stephen

COMPUTATIONAL TECHNIQUES FOR THE EVALUATION OF TOTAL SERVICE TIME IN PACKET-SWITCHED STORE-AND-FORWARD COMMUNICATION NETWORKS: CYCLIC ALLOCATION AND BOUNDING PROCEDURES

City University of New York

PH.D. 1982

COMPUTATIONAL TECHNIQUES FOR THE EVALUATION OF TOTAL SERVICE TIME IN PACKET-SWITCHED STORE-AND-FORWARD COMMUNICATION NETWORKS: CYCLIC ALLOCATION AND BOUNDING PROCEDURES

by

IRA STEPHEN RUDOWSKY

A dissertation submitted to the Graduate Faculty in Engineering and Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1982

The manuscript has been read and accepted for the Graduate Faculty in Engineering and Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

5/9/82
date

Jacob Rootenberg
Chairman of Examining Committee

5/9/82
date

Paul R Kramer
Executive Officer

Professor Jacob Rootenberg, Mentor

Professor Michael Anshel

Professor Peter Kolesar

Professor John Moyne

Professor Pat Sterbenz

Professor Howard Wasserman

Supervisory Committee

The City University of New York

Abstract

COMPUTATIONAL TECHNIQUES FOR THE
EVALUATION OF TOTAL SERVICE TIME IN
PACKET-SWITCHED STORE-AND-FORWARD
COMMUNICATION NETWORKS:
CYCLIC ALLOCATION AND BOUNDING PROCEDURES

by

Ira Stephen Rudowsky

Adviser: Professor Jacob Rootenberg

The research reported herein deals with the problem of computing the total service time of packet-switched messages that are stored-and-forwarded from source node to destination node over computer-communication networks.

Previous results reported on in the literature relied exclusively on the Independence Assumption. The research to be reported on here divorces itself from the Independence Assumption and obtains results regardless of the arrival distribution of messages at the source node. This provides a more realistic solution to the real-world problem.

Two techniques are presented for the computation of total service time--one for an exact solution and one for computing upper and lower bounds. The latter technique, based upon the novel idea of cyclic allocation, complements the former by allowing a quick weeding-out of the infeasible

solutions by testing if the required total service time falls within the bounds. This is done in much less time than needed by the exact solution. As the exact solution is a recursive algorithm whose complexity grows exponentially, the bounding techniques also save time and money for the analyst by allowing him to concentrate only on viable configurations. Thus, two original tools are provided to enhance the process of network analysis and design.

Both techniques have been programmed in interactive PL/I and are included in the appendices of the research.

ACKNOWLEDGMENTS

I would like to gratefully acknowledge the guidance and assistance of my mentor, Professor Jacob Rootenberg, whose support and encouragement brought my goal to fruition.

I would also like to acknowledge the guidance of Professor Michael Anshel, whose advice and friendship I was most happy to receive during my graduate career.

To my mother and late father I am indebted for their love and sacrifices through the years, and thank them for what may have seemed at times to be a thankless task.

I thank my son, Asher, for his cheerful smile and happy chatter that brightened my evenings of research.

Above all, I wish to express my gratitude to my wife, Noga, whose steadfastness and faith in me made it all the more worthwhile.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION AND GENERAL REVIEW

1.1 Historical Notes

During the 1960s, the field of modern data communications was conceived. During the 1970s, birth occurred (Frank, 1979). Computers were originally developed to serve as fast calculating machines, but soon thereafter their usefulness as data processors became evident. Then the potential of computers for facilitating communication became apparent--but it was essential that they be able to communicate easily with people and with other computers. This necessitated the creation of computer communication networks. Thus, a new era was born.

The advent of computer networks in the '60s and the rapid growth of the computer communication area brought about the awareness of the importance of efficient time-sharing schemes to the computer community. Such networks provide centralized computer service for a large number of remote users. The star, loop and tree configurations (see Figures 1.1-1.3) are the most common topologies for centralized networks. The resource computer is located at the root and the terminals are located at the nodes.

Single-application networks were historically the first to appear, examples being the early airline reservations systems such as the American Airlines SABRE system (Perry & Plugge, 1961), the early defense networks such as the

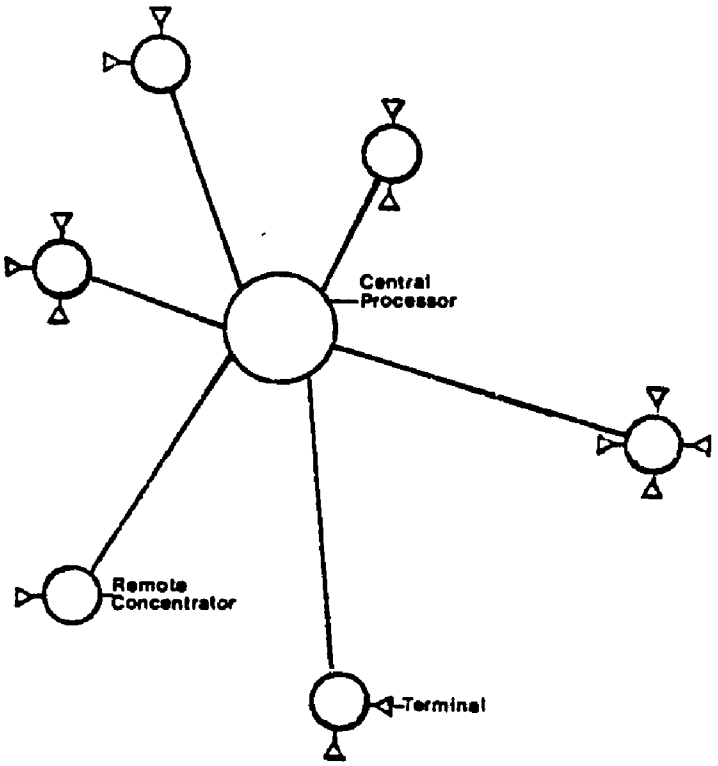

Figure 1.1. Star network

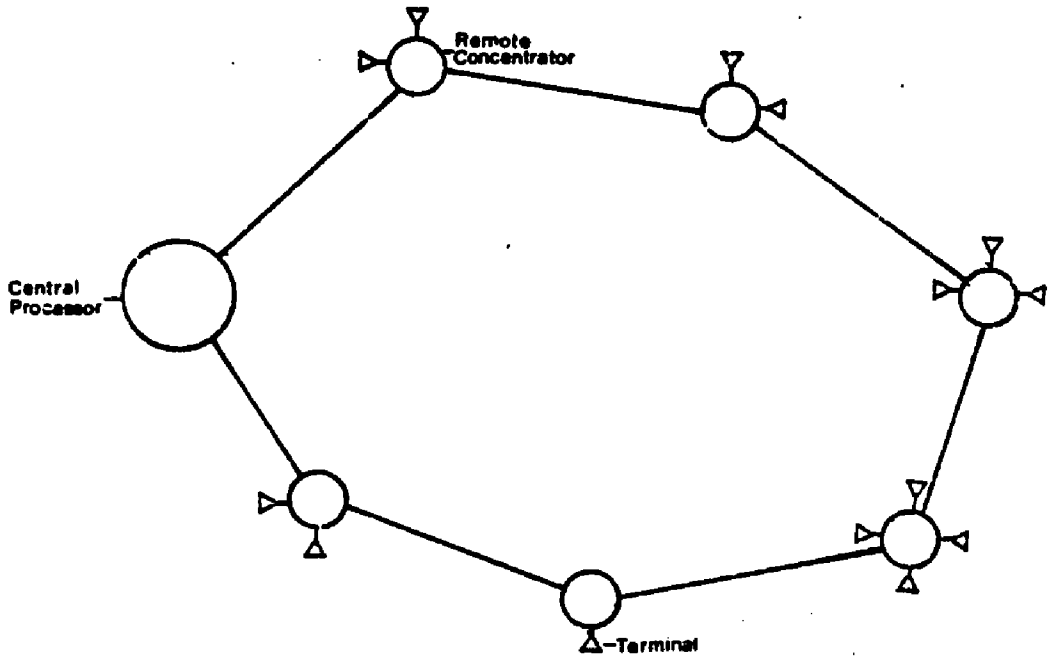

Figure 1.2. Loop network

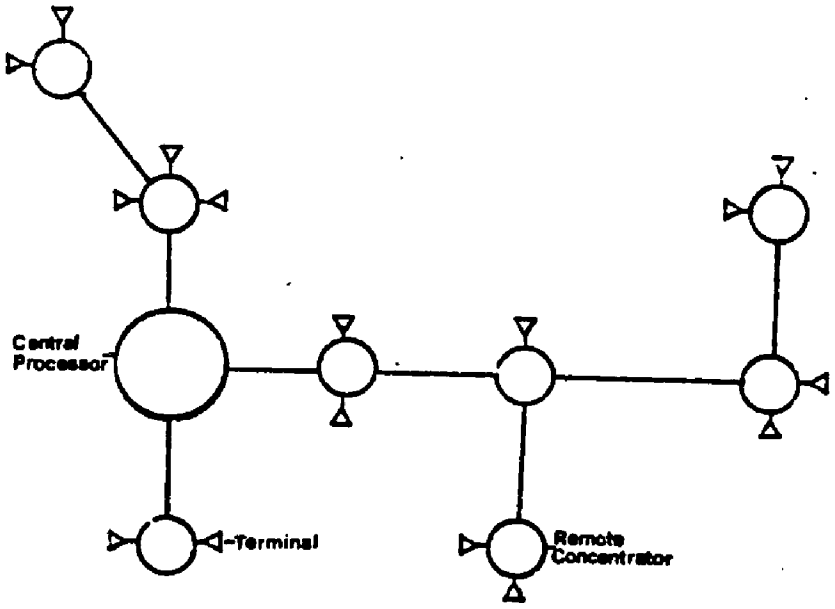

Figure 1.3. Tree network

SAGE (Everett, Zraket & Benington, 1957) and BMEWS systems, and the prototype in-house time sharing nets such as MIT's well-known Project MAC system (Fano, 1965). More general-purpose systems such as the first commercial time sharing networks of the mid-1960s then followed (Morrisey, 1965; Popell, et al., 1966). These systems represented virtually the first instances of using telecommunications to interconnect suppliers of computer services with their customers at remote locations.

In the late 1960s it was recognized that existing computer systems could be more efficiently utilized by connecting them together (Roberts & Wessler, 1970). The special resources and capabilities developed at each of the many separate computer facilities could profit by resource and load sharing. Sharing reduces computer related costs by eliminating redundancy of hardware, software and data bases. This arrangement can be supported by two justifications--one theoretical and one practical. The former is based upon the Law of Large Numbers (Kleinrock, 1975). An individual's demand on the system is bursty, i.e., unpredictable arrival time and size of messages. However, the collective demand of a large population of users is well approximated by the sum of the average demands required by that population. That is, the statistical fluctuations in any individual's demand are smoothed out in the large population case so that the total demand process is more deterministic. The practical justification is based upon the dramatic reduction in the cost of

computer hardware around 1969. This price decrease made it more cost effective to connect into an existing computer network than to buy additional hardware and software for in-house use only. The above-mentioned factors led to the growth of distributed computer-communication networks.

The '70s saw an enormous growth in distributed networks. Banking and insurance firms have extensive networks for electronic funds transfer, credit checking, stock and bond sales and other monetary needs. The medical and law fields maintain extensive on-line centralized libraries for rapid retrieval of information to provide better services. Point-of-sales terminals are used by shopping centers for inventory control to be more in tune with consumer demand. Airline reservations systems, e.g., Société Internationale de Télécommunications Aeronautiques (SITA) (Chretien, König & Rech, 1973), provide to its members worldwide data communications capability for customer service. Commercial time-sharing is available via TYMNET (Schwartz, 1977) and General Electric Information Services MARK III (Schwartz, 1977). The ARPA Network (Walden, 1975), originally conceived by the Advanced Research Projects Agency of the United States Department of Defense, interconnects dissimilar computers throughout the world, thereby allowing users and programs at one computer center to access and interactively use facilities at other centers geographically remote from one another.

With the double-digit rate of increase in the cost of energy, transportation and personnel and the double-digit rate

of decrease in the cost of computing and communications, the demand to bring remote information to users via data networks for cost savings and convenience is overwhelming. By applying the new technology in microelectronics and digital communication creatively to public needs, common user data networks can be expected to bring about a large variety of innovative network services. Future development lies in the adaptation of computer networks for individualized, personal services in distinction to the present services for large organizations.

Leading the field in research and development for advanced computer and communications systems is the Defense Advanced Research Projects Agency (DARPA), which seeks totally new approaches to data acquisition, processing and transmission. One area of special emphasis is computer communications technology.

An installation plan is being developed for ARPANET for use at Strategic Air Command headquarters. In addition, packet radio network technology is being extended to provide automated data processing services for airborne use, such as in the airborne command post. The packet-communication technology program is also exploring computer-based methods for controlling, allocating and accessing information through a variety of media including mobile radio, broadcast satellites, coaxial and optical cable, and leased telephone circuits. End-to-end security is a high priority for these packet-switched networks as well as for packet-switched voice.

Integrated voice and data communications is seen by

both the military and private sectors as the means for future network design. Integration will serve the purpose of network economization as well as network management simplification.

Another area of great potential is "voice mailbox" service. Voice data are stored-and-forwarded from source to destination and saved on disk until the user accesses the system at his convenience. Numerous private companies, among them Datapoint, IBM and ECS Telecommunications, are in the market for providing such systems and the competition will intensify as the market develops.

1.2 Network Structure

A computer communication network is a collection of nodes, i.e., computing resources, connected through nodal switching computers which communicate with each other via a set of links, the data communication channels (Kleinrock, 1964). Messages, in the form of commands, inquiries, file transmissions and the like, travel through this network over the links. At the nodal switching centers, the communication oriented tasks of relaying messages and of inserting and removing them at the proper nodes must be carried out. These tasks are separated from the main computing functions required of the node and are relegated to a switching computer which is dedicated to these tasks.

This separation of tasks, communication and computing, is a historical result of the extension of centralized networks to distributed networks. During the '60s, centralized networks developed an expertise in a particular area, e.g., text

editing, graphics or data base design. For a user of one centralized network to access the programs of another network, the user either traveled to a terminal linked to the desired network or obtained a copy of the software and installed it on the mainframe of his network, assuming hardware and software compatibility of the two systems. If several networks wanted to share their software with each other, numerous copies would have to be produced. Besides this redundancy, in order for one network to be able to run all the software on its computer, a larger and costlier hardware configuration would be required.

The logical course taken to avoid these difficulties was to connect the roots of centralized networks so that only one computer need have a copy of a software package and each user could access any of the other networks from his local terminal. Software maintenance is simplified as any change in a program has to be made only once since all users access the same program. No problems with hardware compatibility arise as the program is run on the machine it was written for. Thus a distributed network allows a user at a terminal to access the needed resources anywhere in the network with the communication process being transparent.

Figure 1.4 depicts the general structure of a distributed computer communication network. The basic elements consist of terminals where all user inquiries originate, remote concentrators to handle traffic in a localized area and send it to a computer resource node (hardware and software configuration). Here the message is processed and forwarded over a

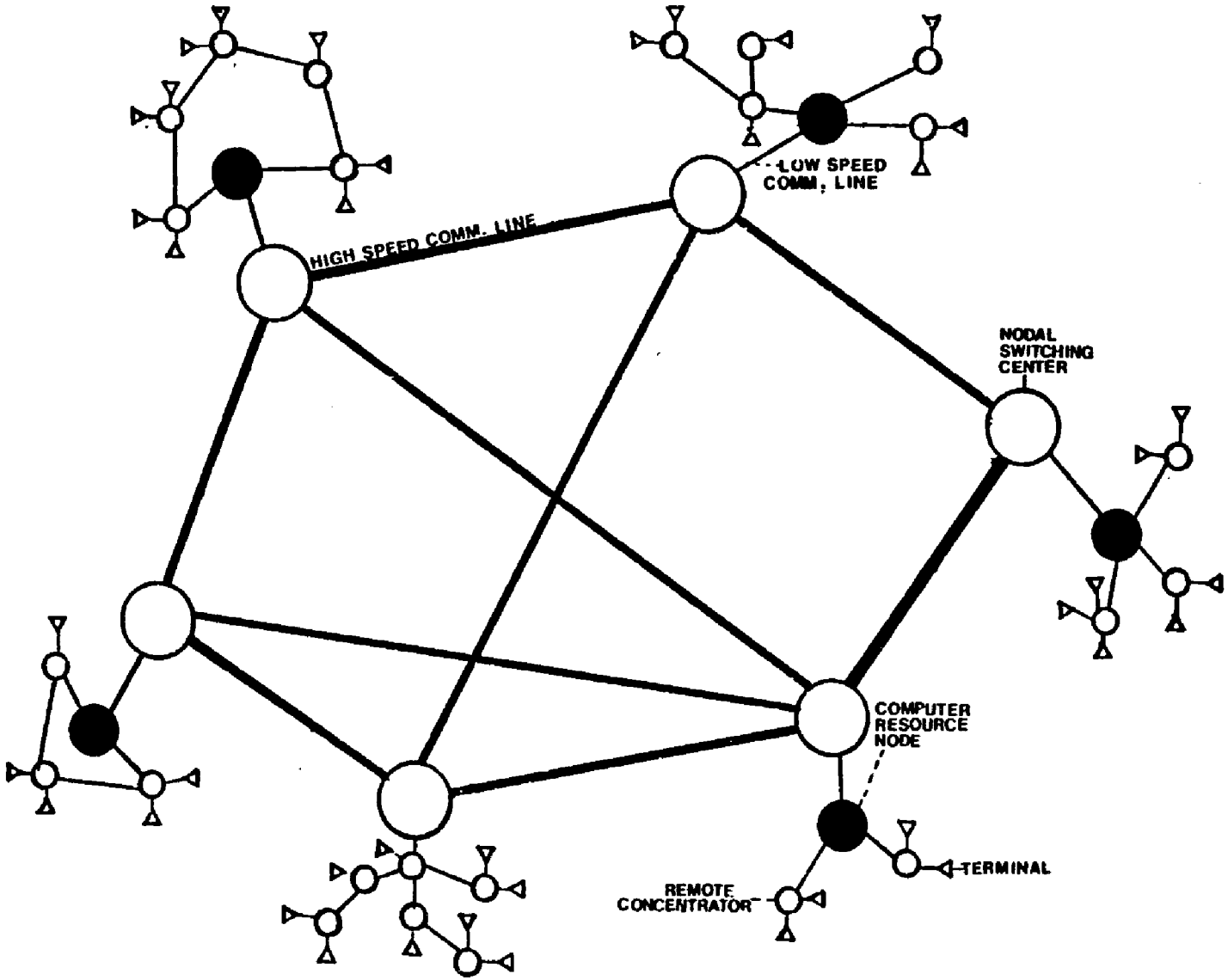

Figure 1.4. General structure of distributed network

low-speed communication line to a nodal switching center where all message handling is taken care of. The proper path is chosen by which it will reach the desired resource node over high speed communication lines. A response to the inquiry will be returned to the terminal of the user from the resource node. Users might be connected to the resource node in either a star, loop or tree configuration which is in turn connected to the switching center, or a user can be directly connected to a switching center. The nodal switching center relieves the resource node of message transmission-related tasks as well as serves as an interface for the various resource nodes which may be of different manufacture. All protocol, error checking, translation and statistics gathering is done at the switching centers as well as the critical task of path selecting.

Computer-communication networks may therefore be partitioned into two separate subnetworks: (a) the communication subnetwork providing message service, and (b) the user-resource subnetwork which is the collection of computer and terminal resources. The communication subnetwork is an area of intensive research in the literature. Four optimization problems, which are presently solved to various degrees of completeness, are being investigated in the literature. They are: (a) capacity assignment; (b) flow assignment; (c) capacity and flow assignment; and (d) topology, capacity and flow assignment. The objective of each problem is to minimize the average message delay, each with respect to a different parameter or set of parameters (as the name

associated with the problem denotes). The proposed research falls under the flow assignment category. For a better understanding of the problem, an explanation of how messages are switched and routed between source and destination will be presented.

1.3 Information Transmission

Upon arrival from a terminal at the local computer resource node, the source node, a message requires a method of transmission and a path in order to reach the destination resource node. The method of transmission, switching technique, is an integral part of the design process as it affects part of the hardware selection. Similarly, how a path is to be determined each time a message arrives at a source node governs, in part, the software as well as the hardware choices. Both of these aspects of information transfer, which are crucial to the ultimate design of the network, will be discussed below.

1.3.1 Switching Methods

Communication networks may be classified by three types of methods for transmission: circuit-switched, message-switched and packet-switched.

A circuit-switched network provides service by setting up a total path of connected links (channels) from the source to the destination. The complete circuit is set up by a special signaling message that threads its way through the network, seizing links in the path as it proceeds. After the

path is established, a return signal informs the source that the phase of actual data transmission may proceed--all channels in the path are then used simultaneously. The entire path remains allocated to the transmission, whether or not data are being sent over it, until the source node releases the circuit. Then, all the seized channels are released and returned to the available pool for use in other paths. Circuit-switching is the common method for telephone systems.

In message-switching only one line is used at a time for a given transmission. The message travels from its source to the first intermediate node in its path. When the entire message is received at this node, a link to the next intermediate node is selected. If all lines to the node are in use, the message is queued in a buffer until the line becomes available and then transmission resumes. Thus, the message "hops" from node to node, using only one channel at a time, queueing at a node if the outgoing links are in use, as it is successively stored-and-forwarded through the network.

Packet-switching is basically the same as message-switching except that each message is decomposed into smaller, fixed size pieces called packets. The packets are numbered and addressed and make their way through the network in a packet-switched store-and-forward fashion. Many packets of the same message may be in transmission simultaneously in the network along different paths; this "pipelining effect" is one of the main advantages of this technique. The transmission delay may be considerably reduced over message-switching as a result.

Digital data transmission, especially when bursty, lends itself very nicely to message and packet switching. The latter can more efficiently handle small messages than message switching in spite of the presence of long messages because of the decomposition process. See Figure 1.5 for a pictorial comparison of these three methods.

1.3.2 Routing Techniques

The second critical decision in network design is how to determine the best path over which to send messages or packets. Deterministic and stochastic routing are the two basic techniques that can be employed.

Deterministic routing is designed to provide satisfactory performance, on the average, over a range of traffic intensities. One method to provide this performance is the least-time delay algorithm (Fultz & Kleinrock, 1971) in which routes are chosen to minimize the overall average time delay. A second approach (Silk, 1969) involves the shortest-route computation using either the path with the fewest number of links or weighing each link based on cost, length, propagation delay, estimated traffic, etc. A central supervisory program establishes a route from source to destination possibly using one of the above methods. The route may be newly determined each time a user comes on the network, remaining unchanged during the session, e.g., TYMNET. Or, the route may be fixed according to predetermined priorities, e.g., in SITA the routes are updated three times a year. In both cases,

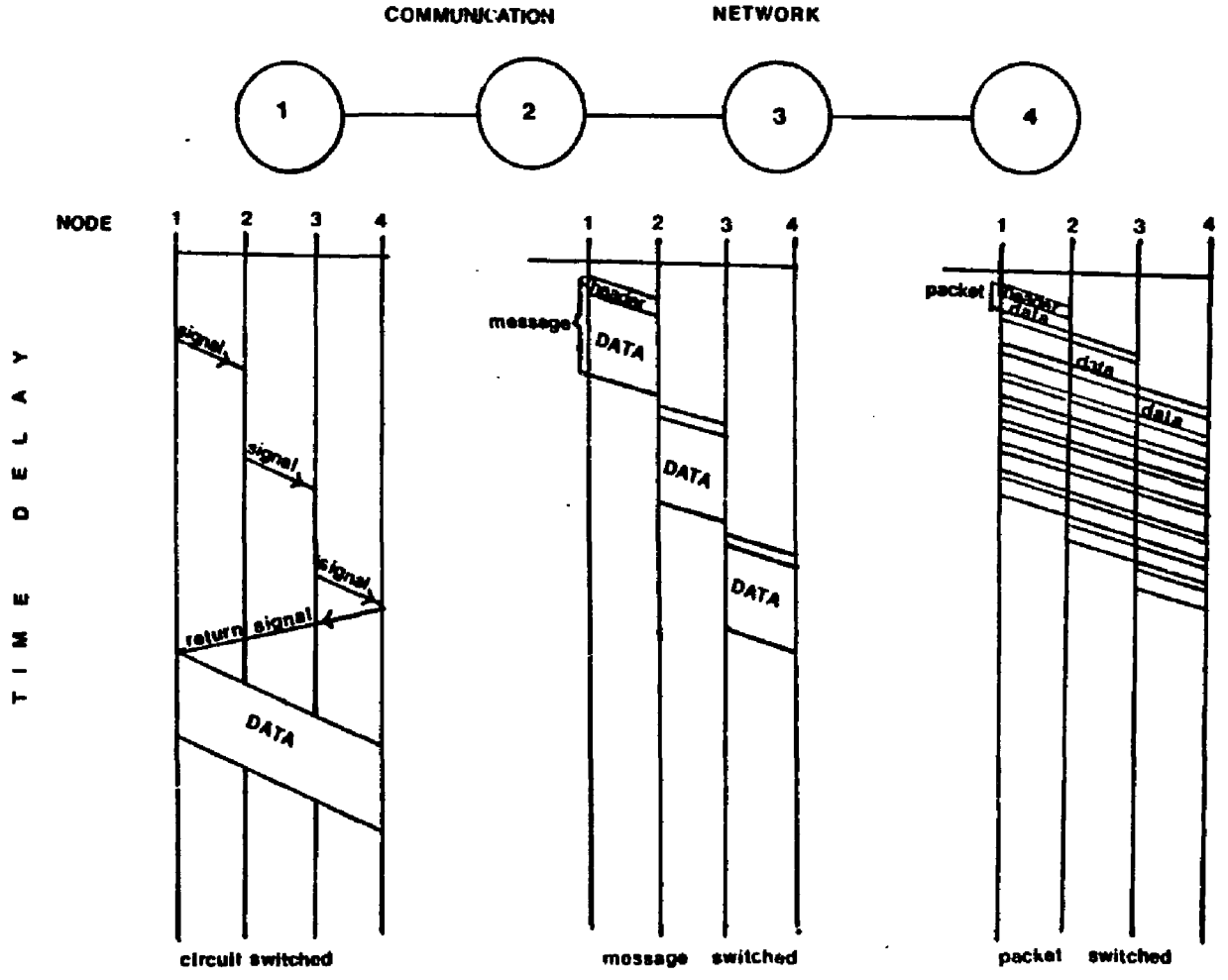

Figure 1.5. Circuit, message and packet switching

secondary routes can be selected to provide better performance, for example a specified fraction of entering messages use the secondary rather than the primary route.

Those procedures in which there are fixed decision rules as to which neighboring node to visit next based upon some probability distribution are known as stochastic (random) techniques. Each node has associated with its outgoing lines a fixed probability as to where a message should be routed next in its journey from source to destination. This random routing based upon probabilities is simple to construct (no source or destination information is required nor is a directory needed), is relatively insensitive to changes in network structure and is highly stable. Kleinrock (1972) has found that for a particular class of random routing procedures, the expected number of steps a message takes can be computed and an expression for the expected time spent in the network can be derived. This technique, however, is highly inefficient and leads to increased time delays. Messages are routed over relatively more links than in a deterministic case. As a result, the increased internal traffic leads to an increase in network overloads. By not taking into account information such as traffic patterns, topology, queueing statistics, etc., poor response time is the consequence (Prosser, 1962).

A more sophisticated and efficient stochastic routing strategy can be found in the decentralized network. The ARPA network, for example, employs this technique whereby each node carries out least-time estimates and decides, on a

decentralized or local basis, which outgoing link to use to minimize the estimated time delay from source to destination.

Isolated local routing techniques allow for each node to make its own routing decisions, independent of the status of neighboring nodes. The time delay estimate a node uses may be the number of messages waiting in the queue for a particular outgoing link or a weighted version of the time delay incurred by messages going in the reverse direction from the destination to the source. Though easy to set up, this technique does not take into account the status of neighboring nodes to improve routing choices and it adapts poorly to damaged networks.

Locally adaptive routing requires that all neighbors exchange, on a periodic basis, their own estimates of the minimum time required to reach a particular destination. The receiving node then adds on its own estimate of the time required to go to each neighbor, essentially the queueing delay on the appropriate outgoing link, to achieve a better estimate of time delay than the isolated technique. This is a more complex strategy to implement as communication is required between nodes but it adapts more easily to node or arc failure.

The analysis of adaptive routing algorithms is extremely complex. Various approaches have met with limited success. Simulation has been relied upon almost exclusively to investigate algorithms and compare different strategies. The basic problem of analysis is that for a distributed network the study of adaptive routing involves the time varying behavior

of a set of interactive queues, work on which is still in its infancy (Kobayashi, 1974). Some analysis of a primitive form of adaptive routing, appropriate to centralized communication networks, has been carried out (Brown, 1975; Brown & Schwartz, 1975).

1.4 The ARPA Network (Kleinrock, 1976; Schwartz, 1977)

The ARPA network has probably generated more interest and excitement in the field of computer networking than any other network in the world. It has spawned a tremendous amount of research in computer-communications analysis and design. The development of the ARPA network has led, directly or indirectly, to the development of a whole host of large-scale computer-communication networks worldwide, both commercial as well as government owned.

In 1967, Roberts (1967) proposed an experimental computer network, which was later to become the United States Department of Defense Advanced Research Projects Agency (ARPA) Network. For a number of years prior to 1967, ARPA had been funding the growth and development of many multiaccess time-shared computer systems at a number of university and industrial research centers across the United States. By 1967 many of these had shown themselves to be valuable computing resources and it was recognized that the Department of Defense as well as the scientific community could benefit greatly if remote access to all these systems could be made available from any terminal. A cost analysis performed at the time indicated

that the use of packet switching for the ARPA network would lead to more economical communications and better overall availability and utilization of resources than many other methods.

The overall network design considerations included: (1) a least-cost network with packet delay of less than two-tenths of a second from source to destination resource node; (2) at least two-connectivity; (3) the ability to respond to variations in traffic flow without significant degradation; (4) the ability to grow in an orderly, low-cost manner.

In 1969 the network began operation with four nodes interconnected; by June 1975 more than one hundred large computers of all types were connected into the system. Packet switching and store-and-forward technology were employed. Nodal minicomputer communication processors called Interface Message Processors (IMPs) carried out the communication functions. IMPs were modified versions of the Honeywell DDP-516 machine. Terminal Interface Message Processors (TIPs) allow terminals to be connected directly to the network without going through a computer system (Host). TIPs (Honeywell H-316 computers) are also used as nodal concentrators. IMPs and TIPs, comprising the communication subnetwork, are interconnected with wideband communication lines primarily of 50 kilobits per second (kbps) capacity. In February 1976 the geographical map of the network consisted of 56 nodes interconnected by 74 links. Two links were 7.2 kbps satellite links. There were 23 TIPs and 33 IMPs. From one to four

Hosts were connected to each IMP and each node is connected to from two to four other nodes to provide alternate path capability.

Messages from Host computers are limited to 8,063 bits to avoid congestion and ease the requirements on nodal buffers. A message from a Host, transmitted to its IMP for processing and forwarding, is segmented at the IMP into at most eight packets with a maximum of 1,008 bits each. In practice the vast majority of messages transmitted are one packet long. Control and destination characters are added at the source IMP before the packet is released to the network. Packets are individually routed through the network to the destination node. Once a complete message is received at the destination IMP and in turn passed on to the appropriate Host, a "ready for next message" (RFNM) packet is returned to the source node.

The ARPA network employs locally determined adaptive routing. Each node makes its own decision as to which node to next forward a given packet. A routing table is maintained at each node which associates the destination nodes with a particular outgoing link estimated to provide the shortest time path. Estimates are obtained by periodically exchanging least-time estimates for each destination node with the adjacent nodes and then a given node adds its own time estimate to get to the neighboring node (based on the queue on each outgoing link). Updates are performed on the routing tables using these numbers. Routing messages are exchanged

at least every 625 milliseconds.

During one week in August 1973, some statistics were gathered on the performance of the ARPA network. The traffic in 1973 was about 4.5×10^6 packets per day, as opposed to 10^5 packets per day in 1971, and appears to have leveled off at this point. Single packet messages made up between 97 and 98 percent of all messages, the average message length being 1.12 packets. The average round-trip message delay (the time a message enters the network until a RFNM is received) was 93 milliseconds, below the .2 second design requirement. This was due to the short length of most messages, low traffic intensity and the very short path messages were found to traverse on the average. Sixteen percent of the traffic traveled one hop, i.e., between neighboring nodes, with the mean path length being 3.3 hops. Most trips followed a shortest path route. The average network arrival rate was 47 messages per second with a peak of 110 messages per second. Including overhead, this corresponds to a utilization of 7 percent of transmission capacity on the average. The maximum utilization was 13.4 percent.

1.5 Literature Review

The flow assignment problem has been defined as the problem of finding the fixed routing policy which minimizes the average delay time (Frank & Chou, 1971). In Frank and Chou (1971) the authors show how the routing problem can be formulated as a variation of the classical multicommodity flow

problem. The expression to be minimized is the sum of a set of convex functions with a convex constraint set. Thus there is a unique local minimum, which is also the global minimum, and it can be found using any downhill search technique. Several optimal techniques for the solution of the multi-commodity flow problem are found in the literature (Danzig, 1963; Tomlin, 1966); however, their direct application to the routing problem proves, in general, to be cumbersome and computationally inefficient. Frank and Chou (1971) present a conceptually simple and computationally efficient heuristic providing a suboptimal routing procedure for minimizing time delay called the minimal link algorithm. Its major drawback is that of being rather insensitive to queueing delays and therefore possibly far from optimum in heavy traffic conditions.

A downhill search algorithm, called the flow deviation method, was developed by Fratta, Gerla and Kleinrock (1973) to compute the optimal solution for the fixed routing problem. The step that is most time-consuming, computation of shortest routes based on the link flow, requires an amount of computation between n^2 and $n^2\log(n)$ where n is the number of nodes in the network; the flow assignment calculation requires on the order of n^2 calculations. The flow deviation provides the total flow in any link. To determine the individual commodity flows in each link, as required for routing assignments, additional bookkeeping must be carried out. One possibility is to update the individual commodity flows at each iteration.

It thus compares favorably to the minimum link algorithm in computation time. However, its rate of convergence is rather slow (Best, 1975).

The flow deviation algorithm has given rise to other algorithms that attempt to improve upon it. The objective function of Frank and Chou (1971) is strictly convex and, as mentioned above, one and only one local minimum exists and it is also the global minimum. In order to find the minimum, the extremal flows method (Cantor & Gerla, 1974), applies the decomposition method (Danzig, 1963) to formulate a master problem with simpler constraints than those in Frank and Chou (1971) but with an enormous number of variables. The master problem is solved first with a restricted set of extremal flows (shortest routes) using the gradient projection method (Hadley, 1964) and then this solution is tested to determine if it is also the optimal solution to the global problem. The gradient-type iteration speeds up the rate of convergence and its computational complexity is comparable to that of the flow deviation method. However, the cost becomes prohibitive for very large nets (more than 100 nodes). A fourth algorithm (Schwartz & Cheung, 1976), for message-switched store-and-forward networks, uses a gradient search modified to account for the constraints of the problem. Its rate of convergence is generally faster than that of the flow deviation algorithm, with the proper initial feasible flows, and it outputs the individual commodity routes directly with no additional book-keeping. Its computational complexity, however, is greater

than that of the flow deviation algorithm. It appears primarily suited to routing computation in small networks or in those networks which have only a subset of nodes communicating with each other.

1.6 Critique of the Current State-of-Art

As can be seen from the above review of the literature, a good deal of work has been done in studying fixed flow routing with some promising results. However, both the heuristics and exact algorithms are deficient in certain aspects. Heuristics provide only sub-optimal solutions, while exact algorithms are computationally complex from the point of view of computer time and storage. During the model building process an algorithm must be run dozens of times to test numerous possible network configurations, straining the budgeted time and money an analyst is allocated.

Direct mathematical analysis of the general communication network and even the simpler class of tandem nets is complex and intractable. Kleinrock (1972) has shown that the source of difficulty is the dependence between the inter-arrival times and the lengths of internal message traffic. One way to break the above dependency and thus hopefully simplifying the mathematical description is to make the following assumption:

> The Independence Assumption: each time a message is received at a node within the net, a new length v is

chosen for this message from the following probability density function:

$$p(v) = \mu e^{-\mu v}$$

This assumption removes the dependency and produces a model which behaves essentially the same as the original model with respect to average message delay. Particularly in light traffic situations and in those where external messages enter in large enough numbers to wash out the dependence of internal messages arriving from other nodes in the network is this assumption true. The Independence Assumption essentially allows us to ignore queueing dependencies; messages are assumed to be independently generated at each node with lengths independently chosen from an exponential distribution. Thus, each channel satisfies the conditions of the single server exponential channel, an M/M/1 queue. This observation coupled with a theorem due to Burke (1956) (the steady-state output of a queue with n channels in parallel, with Poisson arrivals and with lengths chosen independently from an exponential distribution is itself Poisson distributed) allows us to consider each channel (or node) separately in the mathematical analysis.

Algorithms that seek exact solutions to the fixed routing problem include the Independence Assumption as an essential part of their analysis. This assumption allows for the average delay time of a message traveling from source to destination to be expressed in terms of the average flows in

the channels. The result is:

$$T = \sum_{\text{(all arcs i)}} \frac{\lambda_i}{\gamma} T_i$$

where

T = total average delay per message

λ_i = message rate on arc i

γ = total message arrival rate from external sources

T_i = average delay suffered by a message waiting for arc i (transmission, queueing and propagation delay)

As tempting as it is to say that the flow assignment problem has been solved because of the Independence Assumption, the problem is far from finished. Fixed routing problems have been greatly simplified by this assumption, but the introduction of various alternate routing procedures may easily complicate the mathematics once again. In many cases a simulation still has to be relied on for results. In addition, the assumption is valid only under the certain circumstances mentioned earlier. Other cases revert back to the simulation. In all, the Independence Assumption simplifies the mathematical analysis but the exact solutions found in the literature are actually inexact due to the approximations made by the assumption.

1.7 Research Objectives

In light of the above, a method for solving the flow assignment problem with the following characteristics is still lacking from the literature: (1) it is relatively simple in

comparison to present solutions; (2) it is a true-to-life solution rather than an approximation; and (3) it is applicable for various types of arrival and service time distributions.

The objective of the research is to develop an algorithm for computing the exact amount of time a message will require to travel from source to destination in a given network with capacity constraints on the arcs. This total travel, or service, time is composed of two factors: total shipping time and total queueing time. Total shipping time is the sum of the periods during which a message flows from node to node along arcs in the path. The time spent waiting for intermediate arcs to become available is the queueing time.

To be more realistic in analyzing the message flow, the following pattern will be applied. At the source node, a message is divided into fixed size packets. These packets are sent out over the network using either one fixed route or alternate routes as well. As a packet arrives at an intermediate node, the next arc in its path is checked to see if it is available for transmission. If the arc has available capacity for the packet, the packet is forwarded to the next node. When an arc is being used to capacity, the packet queues at the node on a FIFO queue, waiting until the arc becomes available. This represents a true description of what happens to a packet in traveling from source to destination. The goal of the research is to develop a non-simulation procedure for computing the exact amount of service time a message, following the above flow pattern, will incur in a network given the arc

distance and capacity parameters and the size of the message (in terms of packets). This procedure could then be incorporated into an algorithm to compute the total service time over all paths and then select the optimal path to minimize total service time.

Realizing that the network analyst would need to apply this procedure to numerous initial designs, bounding techniques will be developed to aid in the elimination of designs that are out of desired service time ranges. This would allow a more intensive study of the remaining configurations as analyst and computer time and money is saved by reducing the number of designs to be analyzed. Only promising networks would be analyzed for an exact solution producing the best final design in the least possible time.

1.8 Summary of Thesis

The total service time a message incurs from the source node to the destination node in a network is a critical design criterion. The computation of the exact amount of total service time will be analyzed in Chapter 2 by first looking at the simplest type of network, a one-arc two-node network. This analysis will then be extended to a slightly more complicated network, one with two arcs and three nodes. Though the latter network is not much larger or complicated in design than the one-arc two-node network, numerous additional insights can be obtained by studying it. These insights can then be applied to networks with N-1 arcs and N nodes and then

employed in general networks. A recursive algorithm for the computation of the exact total service time will be presented as a result of this analysis.

To aid in the elimination of designs that do not meet the desired service time specifications, bounding techniques will be presented in Chapter 3. From a study of two-arc three-node networks, upper and lower bounding procedures will be derived. Each bound can be computed in linear time so as to be a viable option in the design stage. These bounding techniques will then be analyzed in the context of an N-1-arc N-node network and then extended to general networks so that bounds can be produced for all paths in the network.

Thus, methods will be presented and analyzed to aid in the design stage of network analysis--bounding techniques for a quick elimination of those designs not meeting the required specifications and an algorithm for the computation of the exact total service time in order to achieve the best possible configuration.

CHAPTER 2

THE EXACT SOLUTION: COMPUTATION OF THE EXACT TOTAL SERVICE TIME IN A PACKET-SWITCHED NETWORK

2.0 Introduction

This chapter is devoted to the analysis of how to compute the exact amount of total service time incurred by a message in going from a source to a destination in a network. First, a closed-formula solution is presented for the computation of total service time in a one-arc two-node network. Insight is gained into the mechanics of queueing and service time between nodes. The flow of packets in a two-arc three-node network is then examined via the Cyclic Allocation Algorithm. This algorithm allows for easier and faster calculation of total service time than by a simulation. Speed-up techniques allow for a further reduction in computation time.

Finally, a novel, recursive procedure is presented that computes the exact service time for an N-1-arc N-node network. Its simplicity and flexibility make it a practical and powerful tool for network design.

2.1 One-Arc Two-Node Network (Rudowsky & Rootenberg, 1979)

As mentioned above, the analysis will begin with the study of a one-arc two-node network, as shown in Figure 2.1. A circle (node) with a number inside represents a nodal switching center and the directed arc connecting them represents a communication link. In this case, messages go

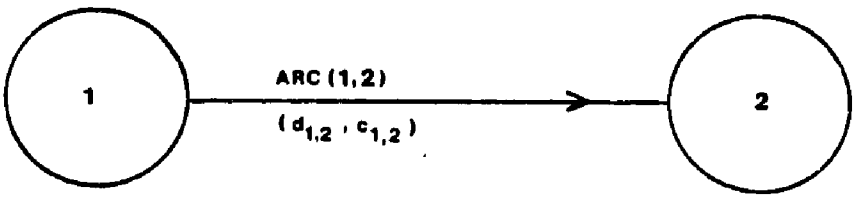

Figure 2.1. One-arc two-node directed network

from node 1 to node 2. Associated with the link is a capacity, $C_{1,2}$ (maximum number of packets that can be transmitted simultaneously between node 1 and node 2), and a distance, $d_{1,2}$ between the nodes (miles, for example). For purposes of analysis, distances will be treated without regard to any particular measure. The time, $t_{1,2}$, to traverse the link is equal to the number of units of distance multiplied by one unit of time per one unit of distance. Thus, for a packet to travel from node 1 to node 2, the distance being ten units, requires 10 units of time. As an example, let $C_{1,2}$ in Figure 2.1 equal 10 packets and $d_{1,2}$ equal 8 units of distance.

The computation of total service time in a one-arc two-node network revolves around the last packet of the message. The time the last packet reaches node 2 is the total service time of the message. Because of the capacity constraints and because there is only one link between source and destination, arc (1,2), packets are sent in blocks. A block consists of at most $C_{1,2}$ packets. If a message is decomposed into 23 packets and the arc capacity is 7 packets, then 4 blocks will be sent; the first three will consist of 7 packets and the fourth block will contain 2 packets. The number of blocks can be determined by computing the ceiling of the number of packets to be shipped, T, divided by the arc capacity, $C_{1,2}$. Each block will contain the maximum number of packets allowable, determined by the capacity. The last block may contain between one and the maximum number of packets.

This value is the remainder obtained from dividing the number of packets by the capacity. If the remainder is zero, then the last block contains the maximum number of packets.

The first block of packets leaves node 1 at time zero and arrives at node 2 at time $t_{1,2}$. The second block leaves node 1 at time $t_{1,2}$ and arrives at node 2 at time $2 \cdot t_{1,2}$. Thus, the kth block arrives at time $k \cdot t_{1,2}$. All that is required in order to compute the service time for the nth packet is to know which block it is in. But this is just

$\left\lceil \frac{n}{C_{1,2}} \right\rceil$, the ceiling of the packet number divided by the capacity. The total service time is then

$$\left\lceil \frac{n}{C_{1,2}} \right\rceil \cdot t_{1,2}.$$

What is most interesting is that the total service time for the nth packet is a function of n. It can be computed independently of the packets that precede it on the link, i.e., the total service time of any preceding packet is not required to be known in order to compute the service time of the nth packet. Given T packets, the total service time is

$$\left\lceil \frac{T}{C_{1,2}} \right\rceil \cdot t_{1,2}.$$

It is important to note that the formula can be applied irrespective of how the messages arrive at node 1, i.e., whatever the arrival distribution is. Messages can arrive Poisson or gamma distributed, following a normal distribution, etc. The only crucial piece of information required to compute the

total service time is how many packets there are in the message. Total service time can be computed assuming an arrival time of zero. Then, by adding on the actual clock time of the arrival, A_t, to the zero based total service time, the actual completion time can be computed. In other words, completion time, C_t, equals
$A_t + \left\lceil \frac{T}{C_{1,2}} \right\rceil \cdot t_{1,2}$.

This first result, though seemingly trivial, provides a basis for analysis of more complicated networks. The concepts of independence of packet flow and the computation of total service time based on the arrival of the last block of packets at the final destination node form an integral part of the study of two-arc three-node networks and eventually of N-1-arc N-node networks. In addition, the formula is not limited to messages only with Poisson arrivals but is applicable to any type of arrival distribution. This is a result not found in the literature as much of the analysis is based upon the Independence Assumption of Kleinrock.

2.2 Two-Arc Three-Node Network

This section will analyze the computation of total service time of messages in a two-arc three-node network. The network in Figure 2.2 will be used to analyze the shipment of a message consisting of 23 packets. Node 1 is the source node, node 3 is the destination, and node 2 is the intermediate node.

For simplicity, assume that the total flow, T, of

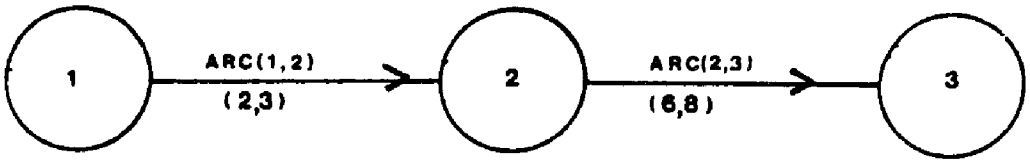

Figure 2.2. Two-arc three-node directed network

23 packets is at node 1 at time zero. As shipping time is part of the total service time, the best strategy is to ship as much as possible at all times. Arc (1,2), with its capacity of three packets, constrains the shipment of flow from node 1 to node 2 into blocks of a maximum of three packets each. The shipping time of any of these blocks along arc (1,2) is two time units. Thus, from time zero to time two, when the first block of three packets is using arc (1,2), the remaining 20 are queued at node 1. Arc (2,3) is idle until time two when the first block of three packets arrives at node 2. At that time, both arcs are active; along arc (1,2) the second block of three packets is flowing from node 1 to node 2 while the first block of three packets is now crossing arc (2,3). The first arc, arc (1,2), is used to capacity but arc (2,3) is not because only three packets have reached node 2 while its capacity is eight packets. But, as much as is available is being sent over it.

At time four, the second block of three packets arrives at node 2 and will immediately be sent over arc (2,3) bringing the total flow over arc (2,3) to six. The third block of three packets from node 1 reaches node 2 at time six. At that time, six packets are on arc (2,3); only two more packets can be accommodated. Until time eight, when the first block of three packets completes its travel over the path and the total packet flow over arc (2,3) becomes five, one packet from the third block queues at node 2. At time eight the fourth block of three packets arrives at node 2. Arc (2,3) has only

five packets flowing over it, but there are now four packets at node 2 to be forwarded to node 3--one queueing from time six and the three new arrivals. Thus, three packets will be shipped and one will queue. This process continues until all packets have reached node 3. Tables 2.1 and 2.2 contain a full account of the message flows over the path in detail and summary form, respectively.

While Table 2.2 provides an event-oriented history of the message flow, Table 2.1 provides a snapshot of the system at each time unit. In Table 2.2 the first column provides the time, unit by unit. The second column contains the total flow shipped from node 1 that has reached node 2 at the specified time unit, while column three contains the amount of flow that has reached the destination node, node 3. The following column is the flow on arc (2,3) and the fifth denotes the queue size at node 2. Note that the sum of the last three columns equals the cumulative flow from node 1 to node 2, i.e., column 2.

Similarly, Figure 2.3 shows a plot of the flow versus time. Starting from time zero, three packets, denoted by ③, are sent over arc (1,2) every two units of time, except for the last shipment which is of 2 packets. As arc (2,3) receives packets at different time intervals, there are three lines to represent the parallel flows. The fourth line depicts the flow that is queued at node 2. For example, a 1 inside a square, [1] means 1 packet is queued.

From these diagrams it is seen that a total of 24 time

Table 2.1

Detail of Flow versus Time

Time Unit	Cumulative flow from node 1 to node 2	Flow at node 3	Flow on arc (2,3)	Flow queued at node 2
0	0	0	0	0
1	0	0	0	0
2	3	0	3	0
3	3	0	3	0
4	6	0	6	0
5	6	0	6	0
6	9	0	8	1
7	9	0	8	1
8	12	3	8	1
9	12	3	8	1
10	15	6	8	1
11	15	6	8	1
12	18	8	8	2
13	18	8	8	2
14	21	11	8	2
15	21	11	8	2
16	23	14	8	1
17	23	14	8	1
18	23	16	7	0
19	23	16	7	0
20	23	19	4	0
21	23	19	4	0
22	23	22	1	0
23	23	22	1	0
24	23	23	0	0

Table 2.2

Summary of Flow versus Time

Time Period	Amount of Flow	Time Period	Amount of Flow
0-2	3	2-8	3
2-4	3	4-10	3
4-6	3	6-12	2
6-8	3	8-14	3
8-10	3	10-16	3
10-12	3	12-18	2
12-14	3	14-20	3
14-16	2	16-22	3
		18-24	1
Total Flow 23 From node 1		Total Flow 23 At node 3	

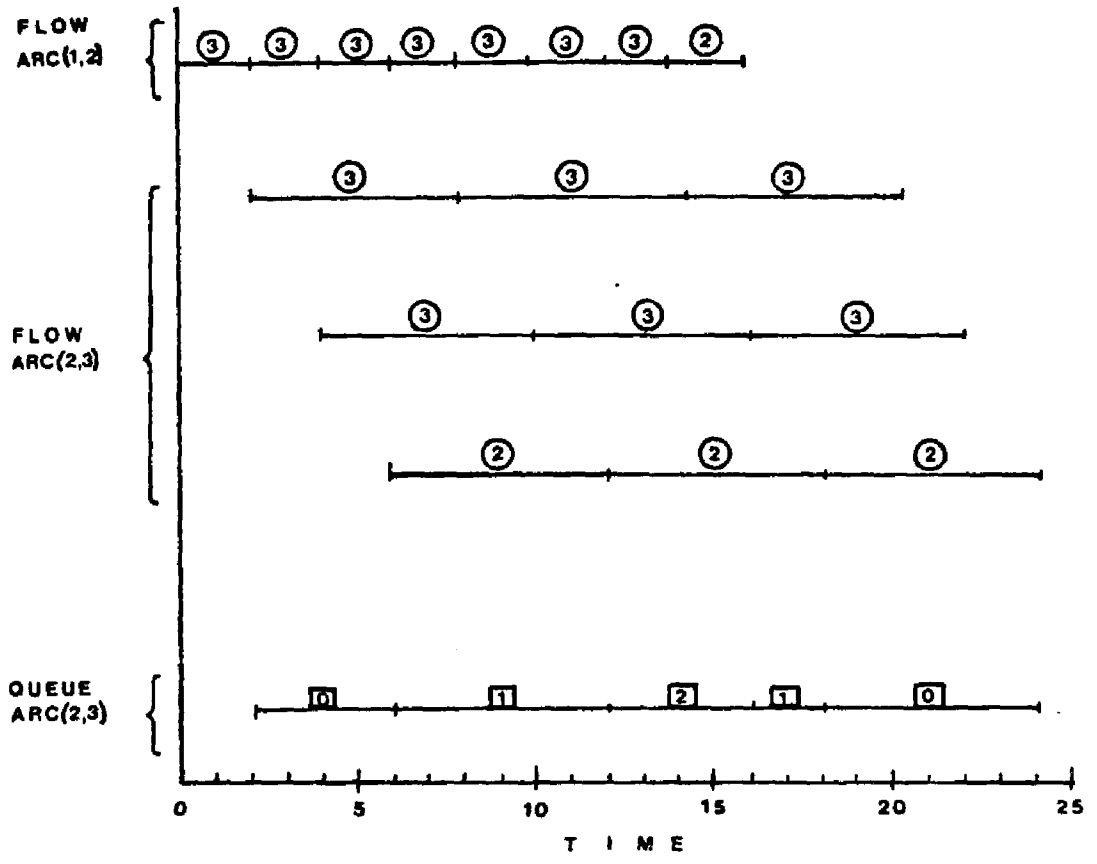

Figure 2.3. Plot of flow versus time

units is required to ship the entire flow of 23 packets from source to destination. Even in this simple example, the book-keeping of how many packets are on each arc is complicated. In effect, Table 2.1 and Figure 2.3 are complete simulations of the flow of the 23 packets from node 1 to node 3. As the number of arcs and nodes in the network increases, so does the complexity of the simulation.

2.2.1 Cyclic Allocation Algorithm

In contrast to this presentation a new approach, which greatly reduces the complexity of the problem, will be introduced. To simplify the computation of allocating flow along arc (2,3), view its capacity of eight packets as meaning that there are eight independent sub-arcs from node 2 to node 3, each with a capacity of one packet. See Figure 2.4 for the new network configuration.

When the first block of three packets arrives at node 2, allocate the first three of the eight sub-arcs, #1, #2 and #3, to carry the packets. The remaining five sub-arcs are idle. The next block of three packets will be sent over sub-arcs #4, #5 and #6, leaving only two of the eight sub-arcs free. At time six, when three more packets arrive, sub-arcs #7, #8 and #1 will be allocated to transport them. Thus, the allocation is done on a cyclic basis, reusing one of the eight sub-arcs only when a cycle has been completed. It is essential for understanding the algorithm to be discussed below that the sub-arcs be selected in the same

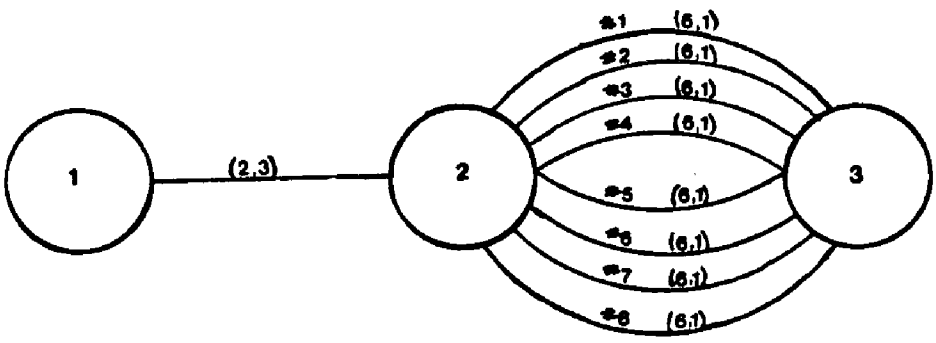

Figure 2.4. New network configuration viewing arc (2,3) as eight independent sub-arcs

sequence, i.e., #1 to #8, returning to #1 only after #8 has been used. Thus, if sub-arcs #1 and #2 are allocated for packets, the next sub-arc to be allocated is #3 even though #1 or #2 may not be in use when #3 is to be used.

Following these rules, the original problem of shipping 23 packets over the path will be solved. There are seven blocks of three packets each and one block of two packets that will flow from node 1 to node 2. This is dictated by the capacity constraint of three packets on arc (1,2) and the total flow of 23 packets; at most three packets may travel over arc (1,2) simultaneously. Twenty-three divided by three yields a quotient of seven, the number of blocks of three packets, and a remainder of two, representing the eighth block of two packets.

To demonstrate the effect of the new approach, the entire allocation procedure will be examined in detail. Later it will be shown how a further reduction in the procedure can be achieved. What follows is a tabular method that shows the assignment of each packet leaving node 1 to a specific one of the eight sub-arcs linking node 2 to node 3. Using the method of cyclically allocating flow to the eight sub-arcs over arc (2,3), the blocks of messages that use each sub-arc can be listed.

Table 2.3 displays the cyclic allocation of the messages by block. The first block, denoted by 1's, uses sub-arcs #1, #2 and #3; the second block denoted by 2's, uses sub-arcs #4, #5 and #6. Sub-arcs #7, #8 and #1 are assigned to

Table 2.3

Blocks Allocated to Sub-Arcs

from Node 2 to Node 3

Sub-arc	#1	#2	#3	#4	#5	#6	#7	#8
Cycle 1	1	1	1	2	2	2	3	3
2	3	4	4	4	5	5	5	6
3	6	6	7	7	7	8	8	

transmit packets of the third block, those with 3's, and so on for the remaining blocks. It should be noted that the last packet of the eighth block travels over sub-arc #7. This same result can be obtained without a full tabular listing. By dividing twenty-three, the total number of packets to be shipped, by eight, the total capacity of all sub-arcs connecting node 2 to node 3, a quotient of two and remainder of seven are obtained. This means that there are two complete cycles of the eight sub-arcs and a third partial cycle that ends on sub-arc #7. Thus, the twenty-third packet is allocated to sub-arc #7. For flows on each sub-arc by packet number, see Table 2.4.

With this information, the next step is to compute the time each packet arrives at node 3, the destination. The arrival time of the twenty-third packet is the total service time required for the entire shipment. The first block of three packets arrives at node 2 at time two, having left node 1 at time zero and traveled two units of distance. It is immediately transferred to arc (2,3) for shipment to node 3 as all sub-arcs to node 3 are free. After six additional time units, the time to travel arc (2,3) being six units, the first block reaches node 3. Thus, sub-arcs #1, #2 and #3 are in use from time two to time eight. The second block of three packets leaves node 1 at time two and arrives at node 2 at time four. It reaches node 3 at time ten using sub-arcs #4, #5 and #6. Block three reaches node 2 at time six. Two of the eight sub-arcs, #7 and #8, are free to be used.

Table 2.4

Flows on Each Sub-arc

by Packet Number

Sub-arc	#1	#2	#3	#4	#5	#6	#7	#8
Cycle 1	1	2	3	4	5	6	7	8
2	9	10	11	12	13	14	15	16
3	17	18	19	20	21	22	23	

However, the third packet in group 3, which is assigned to sub-arc #1 by the cyclic allocation rule, finds sub-arc #1 busy. The first packet of block 1 has not reached its destination at time six when sub-arc #1 is requested for use by packet three of block three; only at time eight will sub-arc #1 be free. Thus, this packet queues at node 2 from time six to time eight.

In a similar fashion, when block four arrives at node 2 at time eight, the first and second packets of that block are allocated to sub-arcs #2 and #3 which are free. The third packet is allocated to sub-arc #4 which is busy until time ten. Thus, this packet queues from time eight to time ten at node 2. Continuing this procedure, the completion time of all flows can be calculated.

The complete computation for all packets in all blocks is shown in Table 2.5. This table is an elaboration of Table 2.3 in that next to each packet, the time of flow over its sub-arc on arc (2,3) is listed in parenthesis. For example, packet 1 of block 1 uses sub-arc #1 from time two to time eight, packet three of block three uses sub-arc #1 from time eight to time fourteen, and packets two and three of block six use sub-arcs #1 and #2, respectively, from time fourteen to time twenty. Note that the cyclic allocation rule forces the starting time of a packet to be no earlier than the completion time of a packet directly above it in a column, as depicted in Table 2.5.

Thus, packet three of block three could not use

Table 2.5

Blocks of Packets with Time of

Flow of Sub-arc of Arc (2,3)

Sub-arc	#1	#2	#3	#4	#5	#6	#7	#8
Cycle 1	1(2-8)	1(2-8)	1(2-8)	2(4-10)	2(4-10)	2(4-10)	3(6-12)	3(6-12)
2	3(8-14)	4(8-14)	4(8-14)	4(10-16)	5(10-16)	5(10-16)	5(12-18)	6(12-18)
3	6(14-20)	6(14-20)	7(14-20)	7(16-22)	7(16-22)	8(16-22)	8(18-24)	

sub-arc #1 at its arrival time at node 2, which was time six, because the previous user, packet one block one, was using it until time eight. Only then, at time eight, could the next packet allocated to sub-arc #1 begin its flow. On the other hand, if packet three block three were to arrive at node 2 at time nine, assuming $C_{1,2}$ was 3 instead of 2, then obviously this packet could not begin its flow over sub-arc #1 until time nine. Though the first packet released its hold on sub-arc #1 at time eight, a packet cannot use a sub-arc if it has not reached the sub-arc.

In total, twenty-three computations must be performed, one for each packet to be shipped. In general, the starting time of a packet allocated to sub-arc #j is the maximum between its arrival time at node 2 and the earliest availability of sub-arc #j. The latter is determined by the completion time of the last packet to use sub-arc #j, i.e., its arrival time at node 3.

2.2.2 Speed-up of the Cyclic Allocation Algorithm

At this point a procedure will be introduced that results in a further reduction of the complexity of the solution to the problem. By dividing arc (2,3) into eight independent sub-arcs, the allocation problem was simplified by using a cyclic assignment scheme. However, twenty-three computations were required. Concentrating attention on the sub-arc that carries the last packet, i.e., sub-arc #7, a reduction in computation is achieved. From the twenty-three calculations previously required, only three are now needed to

achieve the same result.

Because of sub-arc independence, the flows in the other seven sub-arcs do not affect the flow in sub-arc #7. Only those packets that use sub-arc #7 are needed for calculating the total service time. The two questions to be answered are: which blocks use sub-arc #7, and what is the arrival time of each of those blocks at node 2. Answering the first leads to the result of the second.

Numbering the individual packets that use each of the eight sub-arcs (see Table 2.4) by actual packet number rather than block number aids in answering the first question of which blocks use sub-arc #7. From Table 2.4 one can see that the seventh packet is the first to use sub-arc #7. In general, the kth packet is the first to use sub-arc #k. Because of the cyclic allocation of flow and the total capacity of arc (2,3) being 8 units, sub-arc #7 carries each additional eighth packet after the seventh packet. The next packed to use sub-arc #7 is the fifteenth, 7 + 8, followed by the twenty-third packet, 15 + 8, which is the last. Thus, for any of the eight sub-arcs, the k^{th}, $c_{2,3}+k^{th}$, $2 \cdot c_{2,3} + k^{th}$, $3 \cdot c_{2,3}+k^{th}$, . . . and $n \cdot c_{2,3}+k^{th}$ packet travel over sub-arc #k; $n \cdot c_{2,3}+k \leq T$, the total number of packets to be shipped. For sub-arc #7, when n=2 and k=7 we have $n \cdot c_{2,3}+k$ or twenty-three as the last packet to flow over sub-arc #7.

Finding the block each packet is contained in is achieved by dividing the packet number by $c_{1,2}$, the capacity

of the first arc, and taking the ceiling of the result. In our example, $C_{1,2}$ equals three. Thus, packet 7 is in the $\lceil 7/3 \rceil$ block which is the 3rd block, packet 15 is in the $\lceil 15/3 \rceil$ or the fifth block and the last packet, number 23, is in the $\lceil 23/3 \rceil$ or the eighth block. This corresponds to the results in Table 2.5.

With this result we can proceed to the solution of the second question--what is the arrival time of each block using sub-arc #7. The flow from node 1 to node 2 is at a fixed rate of three packets every two time units. Block one arrives at time two, block two at time four, and in general block k at time $k \cdot t_{1,2}$. Once at node 2, a packet will leave for node 3 when a sub-arc is available. The availability of these sub-arcs is not at a constant rate due to the inequalities in the arc parameters, i.e., arc distance and capacity. But knowing the arrival time at node 2 enables one to calculate the completion time of a packet at node 3.

The first packet to use sub-arc #7 arrives at node 2 at time six as it is in the third block ($k=3$ and $t_{1,2}=2$, thus $k \cdot t_{1,2}=6$ as described in the preceding paragraph). Being the first packet to use sub-arc #7, there is no queueing time incurred. It is immediately shipped and arrives at time twelve at node 3. The arrival time at node 3 is the sum of the departure time from node 2, six units in this case, and $t_{2,3}$, the time to travel between node 2 and node 3, which is six time units. Thus, sub-arc #7 will be in use from time six to time twelve. The next packet to use sub-arc #7, packet number fifteen, arrives at time ten, being in the fifth block.

It must wait until time twelve when sub-arc #7 becomes free to continue the journey. Its departure time from node 2 is twelve, though it arrived at time ten, and its arrival time at node 3 is time eighteen, 12 + 6. The determining factor as to when a packet leaves node 2 is the selection of the maximum between the arrival time at node 2 and the completion time of the last packet to use the same sub-arc, i.e., the arrival time at node 3 of the last packet frees the sub-arc for further use. The last packet to use sub-arc #7, packet twenty-three, arrives at node 2 at time sixteen; in the eighth block. As sub-arc #7 is being used by packet fifteen until time eighteen, packet twenty-three queues from time sixteen to time eighteen. It arrives at node 3 at time twenty-four, 18 + 6. Note that its departure time from node 2 was the maximum of eighteen and sixteen.

Having found a method to compute the block number and arrival time at node 2 of all packets using the terminal sub-arc, in this case sub-arc #7, the problem and its solution can be stated as follows:

Problem: Given the network in Figure 2.5, where

$d_{i,j}$ is the distance from node i to node j,

$C_{i,j}$ is the capacity of arc (i,j) and

$t_{i,j}$ is the time to cross arc (i,j),

compute the total service time required for T packets to be stored-and-forwarded from node 1 to node 3 where packets are shipped as soon as sub-arcs are free. For simplicity assume $t_{i,j} = C_{i,j} \cdot$(1 unit time/1 unit distance).

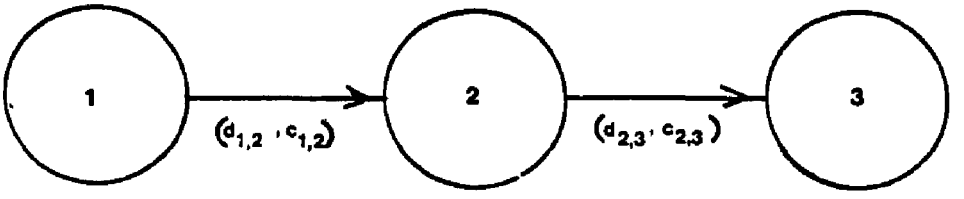

Figure 2.5. General two-arc three-node directed network

Solution: Cyclic Allocation Algorithm for a two-arc three-node directed network.

Define T_k = arrival time, at node 3, of the kth block of $C_{1,2}$ packets that uses the terminal sub-arc, and

S_k = arrival time, at node 2, of the kth block of $C_{1,2}$ packets that uses the terminal sub-arc.

(1) Compute: $R = T \text{ MOD } (C_{2,3})$

If T MOD $(C_{2,3})$ equals 0 then $R = C_{2,3}$.

$Q = (T-R)/C_{2,3}$

(Q will be used in later steps of this algorithm.)

(2) $T_1 = t_{1,2} \cdot \lceil R/C_{1,2} \rceil + t_{2,3}$

(3) Repeat for k = 1 to Q

(3a) $S_{k+1} = t_{1,2} \cdot \lceil (R+k \cdot C_{2,3})/C_{1,2} \rceil$

(3b) $T_{k+1} = \text{Max } (S_{k+1}, T_k) + t_{2,3}$

(3c) return to step 3

(4) T_{Q+1} is the total service time required

Explanation of the algorithm:

Step (1) computes the sub-arc on which the packet flow will terminate, call it R.

Step (2) computes the arrival time at node 3 of the first packet that uses sub-arc R

$\lceil R/C_{1,2} \rceil$ is the block

$t_{1,2} \cdot \lceil R/C_{1,2} \rceil$ is the arrival time at node 2

$t_{1,2} \cdot \lceil R/C_{1,2} \rceil + t_{2,3}$ is the arrival time at node 2 + traveling time to node 3 from node 2.

Step (3) begins the loop for the other packets that use sub-arc R;

Q was calculated in step (1).

Step (3a) computes the arrival time at node 2 of the kth packet to use sub-arc R where $\lceil (R+k \cdot C_{2,3})/C_{1,2} \rceil$ gives the group number of this packet.

Step (3b) selects the maximum of the arrival time at node 2 of the $(k+1)^{st}$ packet, denoted S_{k+1}, and the arrival time at the terminal node of the kth packet, denoted T_k, i.e., the earliest time sub-arc R is available for the $(k+1)^{st}$ packet. Adding $t_{2,3}$ to the maximum results in the earliest arrival time, at node 3, of the $(k+1)^{st}$ packet.

Step (3c) repeats the above calculations for all packets flowing over sub-arc R.

Step (4) the last value of T, T_{Q+1}, is the earliest arrival time at node 3 of the last packet or equivalently the total service time.

Illustration of the Cyclic Allocation Algorithm

The graph in Figure 2.6 will be used as input for the cyclic allocation algorithm. Let T, the total number of packets to be stored-and-forwarded, be 27. Arc lengths and capacities are found in Figure 2.6. Following the algorithm we find:

(1) Q = 6 R = 3

(2) $T_1 = 6$

The computation of S_k and T_{k+1} will be repeated for values of k from 1 to Q where Q was calculated in (1) as 6. The results of the iteration are thus:

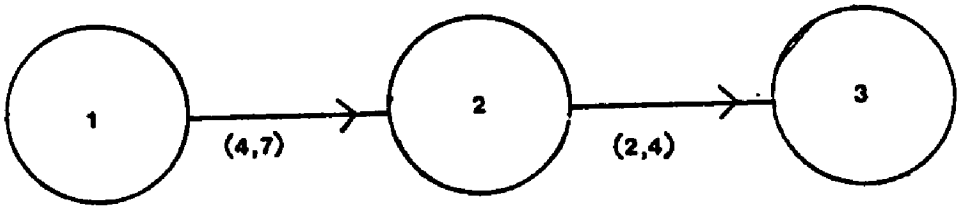

Figure 2.6. Two-arc three-node network to illustrate the CAA

(3) $S_2 = 4$, $T_2 = \text{Max } (4,6) + 2=8$

(4) $S_3 = 8$, $T_3 = \text{Max } (8,8) + 2=10$

(5) $S_4 = 12$, $T_4 = \text{Max } (12,10) + 2=14$

(6) $S_5 = 12$, $T_5 = \text{Max } (12,14) + 2=16$

(7) $S_6 = 16$, $T_6 = \text{Max } (16,16) + 2=18$

(8) $S_7 = 16$, $T_7 = \text{Max } (16,18) + 2=20$

(9) Total Service Time = 20 time units, i.e., T_7

The efficiency of the algorithm is clearly demonstrated over a complete simulation of the message flow that would have been required if not for the findings of this section. Again, note that the Independence Assumption was not invoked in order to justify the Cyclic Allocation Algorithm. Messages can arrive at the source via any distribution and follow a true-to-life flow pattern in going from source to destination.

2.3 N-1-Arc N-Node Network

This section will present the generalization of the Cyclic Allocation Algorithm (CAA) for a network with N-1 arcs and N nodes and its applicability to a general network. A novel method of computation of total service time will be analyzed and described both algorithmically and graphically. In comparison to simulation, the current method for obtaining an exact, true-to-life solution, this new technique will speed up the analysis stage. A simulation would probably require changes in its program code and computer memory requirements in order to accommodate varying test designs. The algorithm to be presented does not require any coding changes in order

to examine different configurations and computer memory is dynamically allocated based upon the network design parameters. This relieves the analyst of programming chores enabling him to concentrate on analysis. In addition, selective rather than exhaustive calculations and parallel computations pointed out by an easily constructed graphical representation speed up the algorithm for a faster, final solution. The key is to use the CAA to center on those packets that are necessary and sufficient to arrive at the value for total service time in the network.

This novel approach will provide the network analyst with an invaluable tool for selecting the best configurations for the network design problem under consideration. Used alone or in conjunction with other design tools, it can speed up the analysis and help to achieve an optimal or near-optimal final design.

2.3.1 General Cyclic Allocation Algorithm

The Cyclic Allocation Algorithm provides a quick and exact method for the computation of total service time in a two-arc three-node packet switched communication network. The key to the algorithm, as seen in section 2.2, is to trace the packets that flow over the terminal sub-arc that carries the final packet. All other packets have no effect on the outcome. The extension of the algorithm to a path with N-1 arcs and N nodes follows the same logic. Start at the last arc in the path and find all packets that flow over the terminal sub-arc. To compute the completion time of packet j

on the terminal sub-arc of arc (i_{N-1},i_N), see Figure 2.7, one must know: (a) the completion time of packet j on the preceding arc, i.e., arc (i_{N-2}, i_{N-1}), (b) the completion time of the last packet prior to packet j to use the terminal sub-arc of arc (i_{N-1},i_N), and (c) the time required to traverse the terminal sub-arc of arc (i_{N-1}, i_N). The larger of (a) and (b) added to the value of (c) yields the completion time of packet j on arc (i_{N-1},i_N). The values of (a) and (b), necessary for the computation, are themselves computed in a similar fashion. Thus, the algorithm can be expressed recursively providing the analyst with a compact tool for design.

For a better understanding and appreciation of the Generalized Cyclic Allocation Algorithm (GCAA), as this new tool will be referred to, an example will be presented and extensively analyzed comparing the unwieldy simulation technique with this simple and faster novel approach. This recursive technique will be shown as well as a unique graphic representation which simplifies the computation and speeds up the algorithm.

Consider the path given in Figure 2.8. Each arc is labeled with a pair of numbers. The first of the pair is the distance and the second is the capacity. Thus, for example, the distance of arc (4,7) is 5 units, denoted $d_{4,7}=5$, and the capacity of arc (7,9) is 3 units, $C_{7,9}=3$. Given that there is a message consisting of 14 packets at node 1, the analyst would want to know how much time is required for all 14

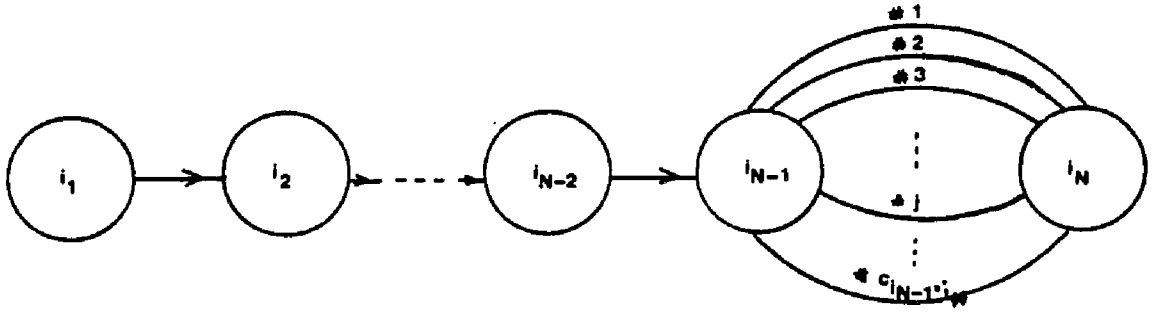

Figure 2.7. N-1-arc N-node directed network with last arc shown as a sequence of independent sub-arcs

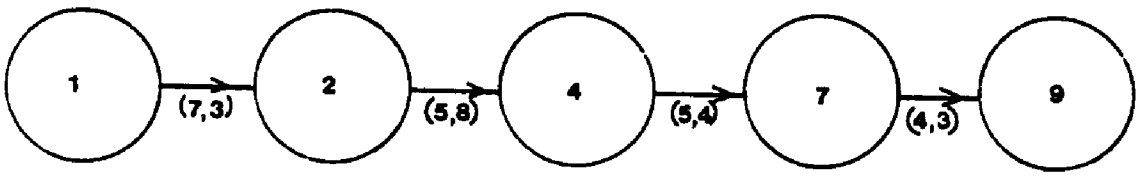

Figure 2.8. Four-arc five-node directed network

packets to arrive at node 9. The following will present and compare two methods of determining the total service time.

Method 1 - Simulation

Starting at the source node, node 1, compute the time each of the 14 packets reaches node 2, i.e., the service time. These values can be calculated from the formula presented in section 2.1. The queueing time of the nth packet on arc (i,j), $q^n_{i,j}$, is

$$q^n_{i,j} = \left\lceil \frac{n-C_{i,j}}{C_{i,j}} \right\rceil \cdot t_{i,j}$$

where $t_{i,j}$ is the time required to ship one packet along arc (i,j). The service time of the nth packet, $s^n_{i,j}$, is then $s^n_{i,j} = q^n_{i,j} + t_{i,j}$. In total, this step requires 14 computations and 14 units of storage, one for each value of $s^n_{i,j}$.

Proceeding to arc (2,4) compute the service time of each packet on the arc and again store these values. The computation of service time from this point on is a selection of the maximum between the arrival time of the nth packet at node 2 and the earliest time the arc is free to transmit the nth packet. Then $d_{2,4}$ is added to the maximum value of the above two numbers and the service time is thus computed. For example, packet 2 arrives at node 2 at time seven and finds excess capacity available on arc (2,4). The service time for packet 2 is thus twelve, obtained by adding the value of $d_{2,4}$ to seven. Similarly, the service time of packet 14 is computed

as forty, 35 + 5, since when it arrives at node 2 the arc is not being used to capacity. If when packet 14 arrived, the arc was not available until time forty, for example, then the service time for packet 14 would be forty-five max (35,40) + 5. For the remaining arcs in the path similar type calculations must be performed until the last packet arrives at the destination node. The service time of the last packet on the last arc is the total service time for the path.

A simulation of this type is very complex and time consuming as each packet must be followed on each arc. In addition, the service time of each packet on each arc must be saved. As the number of arcs and packets grows, the space and time complexity of the simulation increases even faster, making it an unwieldy tool for analysis.

Method 2 - Generalized Cyclic Allocation Algorithm

The key to this method is to follow the minimum number of packets necessary to compute the total service time in the path. Using the method of analyzing an arc as a set of independent sub-arcs, Table 2.6 presents the flow of each packet over each sub-arc.

The algorithm to be presented can be described as follows. Begin at the terminal arc and compute the service time for packet 14 on arc (7,9)--the notation for this will be $p_{14}^{(7,9)}$. Based upon the analysis presented in Method 1, we have that $p_{14}^{(7,9)} = \max (p_{14}^{(4,7)}, p_{11}^{(7,9)}) + d_{7,9}$ where $p_{14}^{(4,7)}$ is the completion time of packet 14 on the preceding

Table 2.6

Packet Flow Over Sub-Arcs:
Circled Packets Are Key Calculations

Sub-Arc	(1,2) 1	2	3	(2,4) 1	2	3	4	5	6	7	8	(4,7) 1	2	3	4	(7,9) 1	2	3
packet #	(1)	(2)	(3)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(1)	(2)	(3)	(4)	1	(2)	3
	(4)	(5)	(6)	9	(10)	(11)	12	13	(14)			(5)	(6)	(7)	(8)	4	(5)	6
	(7)	(8)	9									9	(10)	(11)	12	7	(8)	9
	(10)	(11)	12									13	(14)			10	(11)	12
	13	(14)														13	(14)	

arc, arc (4,7), $P_{11}^{(7,9)}$ is the completion time of the last packet, packet 11, to use the same sub-arc before packet 14 on arc (7,9) and $d_{7,9}$ is the time required to traverse arc (7,9). As $P_{14}^{(4,7)}$ and $P_{11}^{(7,9)}$ are not known, they must be computed before $P_{14}^{(7,9)}$ can be evaluated. This means we must calculate $P_{14}^{(4,7)} = \max (P_{14}^{(2,4)}, \quad P_{10}^{(4,7)}), + d_{4,7}$ and $P_{11}^{(7,9)} = \max (P_{11}^{(4,7)}, P_{8}^{(7,9)}) + d_{7,9}$. The algorithm iterates backwards until a computation of the form $P_{n}^{(1,2)}$ is reached in which case $P_{n}^{(1,2)} = \left\lceil \frac{n}{c_{1,2}} \right\rceil \cdot d_{1,2}$. Arc (1,2) is the stopping point because node 1 is the source node. The value of $P_{n}^{(i_{k-1},i_k)}$ is equal to 0 if n equals 0 or less; this means that there is no such packet on that arc, thus its service time is 0. Whenever a point is reached where an absolute value can be computed, the algorithm traces back to the previous step and continues. In the above example, when values have been computed for $P_{14}^{(4,7)}$ and $P_{11}^{(7,9)}$ then $P_{14}^{(7,9)}$ can be computed and the algorithm is successfully terminated.

The circled packets in Table 2.6 point out those packets that are necessary and sufficient for the computation of $P_{14}^{(7,9)}$ as described in the above algorithm. Table 2.6 also serves as a guide for the computation process by

illustrating the packets and sub-arcs needed for the computation. One can see directly from the table that packet 11 precedes packet 14 on arc (7,9) and that to compute $P_{14}^{(4,7)}$ the service times of packets 2, 6 and 10 must be computed first. Of course, these facts can be generated algorithmically but this table gives an insight into the procedure.

The recursive nature of the algorithm for the computation of total service time in a packet switched network can be better seen through the following psuedo-code:

Algorithm GCAA

GCAA (T,N):

If N-1 = "source node" then

return $\left\lceil \frac{T}{C_{i_{N-1},i_N}} \right\rceil \cdot d_{i_{N-1},i_N}$

Else if $T - C_{i_{N-1},i_N} < 1$ then return

$(GCAA\ (T,N-1) + d_{i_{N-1},i_N})$

Else return $(Max(GCAA(T,N-1),\ GCAA(T-C_{i_{N-1},i_N},N)) + d_{i_{N-1},i_N})$

End GCAA

The name of the algorithm is GCAA and its two parameters are T, a packet number, and N, a node. Initially, the first call to the algorithm is made with T set to the total number of packets to be shipped and N set to the destination node. Then the value of T is checked to be less than or equal to 0, in which case the value of 0 is returned. If T is

greater than 0, it then checks if N-1 is a source node, in which case the service time can be directly computed and returned. If neither of the above conditions is true, then two more computations are required, the service time of the same packet on the preceding arc and the service time of the preceding packet on the same sub-arc. The algorithm computes these values recursively by branching until a terminal computation is reached (i.e., one of the first two conditions is satisfied). Then it backtracks until the initial computation is completed.

2.3.2 Advantages and Benefits of the GCAA

Apparent outright is the simplicity and adaptability of the GCAA over a simulation. A program to simulate the flow of packets over a path would require many more lines of programming than the algorithm presented above. The service time of each packet on each arc would have to be saved during execution of a simulation program, some of the calculations being superfluous. On the other hand, the recursive algorithm remains the same no matter how many nodes in the path and only performs the minimum number of service time calculations necessary to solve the problem. Additionally, no data storage worries need bother the programmer as the recursive calls dynamically allocate and release the exact amount of computer memory needed. To get an idea of the complexity of the algorithm, the call with the maximum backtracking depth can be computed as follows:

find $D = \max \left\lceil \frac{T}{C_{i_{k-1},i_k}} \right\rceil$ $k = N, N-1, N-2, \ldots, 3,2,$

set K equal to the value of k for which D is at its maximum and compute max depth = D+ K-2. (Equation 2.1)

If N equals 2, then there is only one arc in the path and only one computation, i.e., $s^T_{i_1,i_2}$, is needed so the maximum depth is 1. Here T denotes the total number of packets to be shipped. The path in Figure 2.8 has a maximum depth of 8, as can be seen by tracing packet 14 up sub-arc 2 on arc (7,9) and then tracing packet 2 to arc (1,2) where it can be computed directly. Basically what equation 2.1 does is to follow the terminal packet up each sub-arc and back to the source. The maximum number of steps is the maximum depth.

By exploiting the structure of the recursive calls of the algorithm, two benefits can be derived that will speed up the computation. Figure 2.9 depicts the tree created by the branching of the recursive algorithm. Starting with packet 14 on the top diagonal, representing arc (7,9), one sees that to compute the service time of the calling packet, $P^{(7,9)}_{14}$, the service times of packet 11 on arc (7,9) and of packet 14 on arc (4,7) are needed. Similarly, each of these packets requires the service times of two other packets; one, of the packet on the diagonal of the calling packet (the packet number is computed by subtracting the capacity of the arc, noted on the right-hand side of Figure 2.9) and the other on the diagonal below the calling packet (the packet number is the same as the calling one). The process continues until

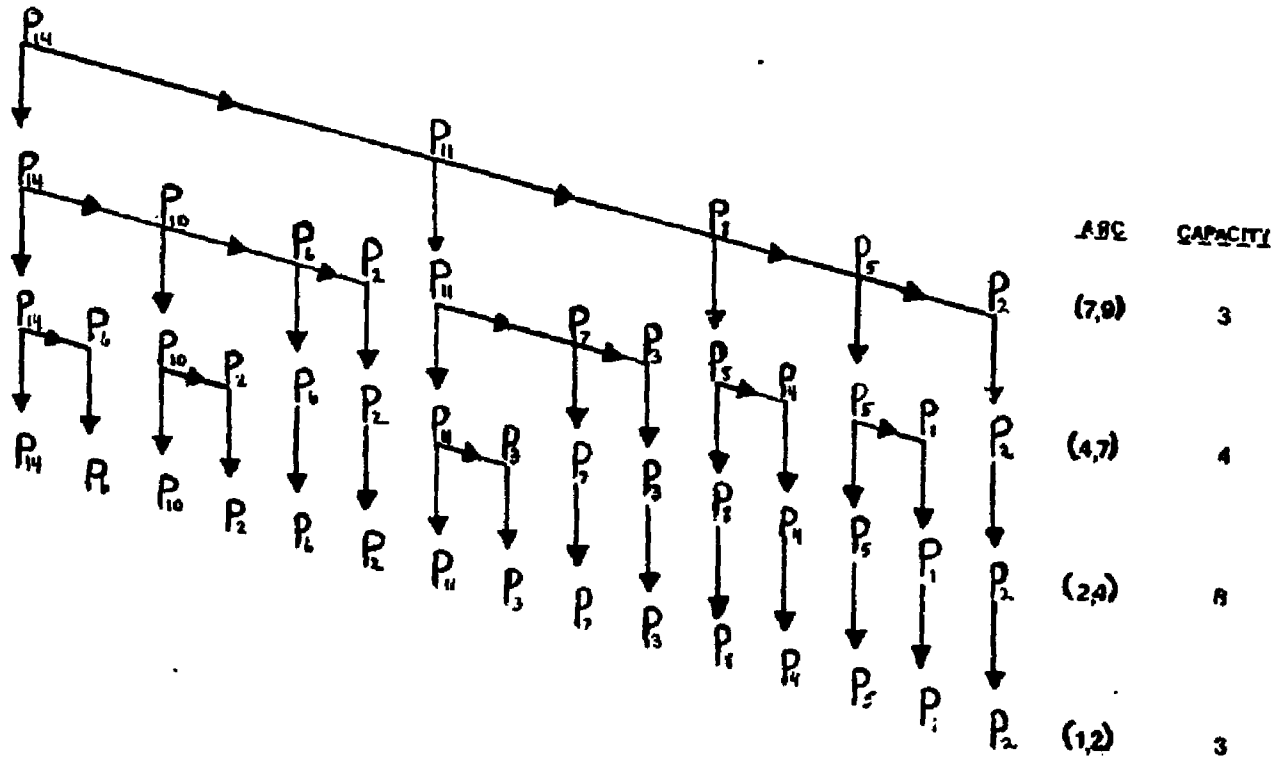

Figure 2.9. Graphic representation of the GCAA

the starting arc is reached in which case no further branching is done.

The first benefit to be exploited is that all computations on arc (1,2) can be done in parallel since they are independent of each other. The second benefit is that those computations that are duplicated in the tree can be done once and the results stored for future reference. For example, the calculation of $P_6^{(2,4)}$ appears twice, $P_2^{(2,4)}$ occurs three times, $P_2^{(4,7)}$ twice, etc. By scanning the bottom diagonal, the duplicate computations can be noted since, if a packet appears more than once on the bottom line, it will be duplicated on the next level and possibly on the level above that, and so on. Thus, the selective computation provided by the GCAA can now be refined to further reduce the number of computations. Comparing the basic two methods along with the refinement by their respective number of computations, the results are: (1) simulation - 56; (2) GCAA - 47; (3) GCAA with refinements - 38. The refinements of using parallel computation and avoiding duplicating calculations speed up the algorithm to provide a more useful and practical tool for the analyst.

A method to implement the enhancement of selective computation involves constructing a linked list for each arc in the path. As the algorithm determines the packets needed for the computation of total service time, that packet is added to the appropriate linked list. Pointers are also set up to the right and left sons of the current

root node.

For example, when packet 14 on arc (7,9) is to be computed, the following elements are added to linked lists: packet 14 to linked list (7,9) and linked list (4,7) and packet 11 to linked list (7,9). Packet 14 on (7,9) will have pointers to the other two entries so that it can be computed after they are. Packet 2 on arc (4,7) appears twice in Figure 2.10 but need only appear once in the linked list and it will thus be computed only once. Similarly, those packets that are needed to compute it need only appear once, i.e., packet 2 on arcs (2,4) and (1,2).

In terms of implementing these linked lists in a computer program, the main concern is with the growth and thus search time of each list. But this is relative to the total computations of the direct simulation for the same network and is thus difficult to quantify in general terms. However, without these enhancements the GCAA tends to be conservative in its computation time because of the duplication of calculations, though this is not a steadfast rule. This conservative aspect is alleviated with the enhancements to the GCAA.

2.3.3 Graphic Representation

The graphic representation of the GCAA algorithm, as shown in Figure 2.9, is very simple to construct. The basic information needed for the construction is the total number of packets to be shipped and the capacities of each arc. Draw

a diagonal for each arc, label it with the arc and its capacity. The top diagonal represents the arc with the destination node, the next lower level is the one preceding it, and so on until the arc containing the source node is the bottom level. At the extreme left of the top diagonal place P_n where n is the total number of packets to be shipped. Directly underneath this, on the next level down, put P_n and continue until the bottom level. Going up one level from the bottom, subtract the capacity of the corresponding arc from the packet number on that diagonal. If the result is greater than 0, place that value on the diagonal to the right of the previous packet number, e.g., from $P_{14}^{(1,2)}$ go up to $P_{14}^{(2,4)}$ and subtract 8, the capacity of arc (2,4) from 14, giving $P_6^{(2,4)}$. Repeat the previous process by going down to the next level and putting a P_6 there until bottom is reached. Then trace back up again. If a subtraction cannot be performed on one level, go up to the next level. The construction stops when a subtraction on the top diagonal will yield a value less than 1. Figure 2.9 is completed with $P_2^{(7,9)}$ because 3, the capacity of arc (7,9), subtracted from 2 yields a value less than 1. Then the scanning for duplicate calculations can be performed as well as the storing of partial computations to speed up the algorithm.

The actual full computation is done by a post-order traversal of the binary tree (see Figure 2.10). Starting at the root of the tree, $P_{14}^{(7,9)}$, traverse its left subtree, then the right subtree, and then visit the root. In Figure 2.10, start at

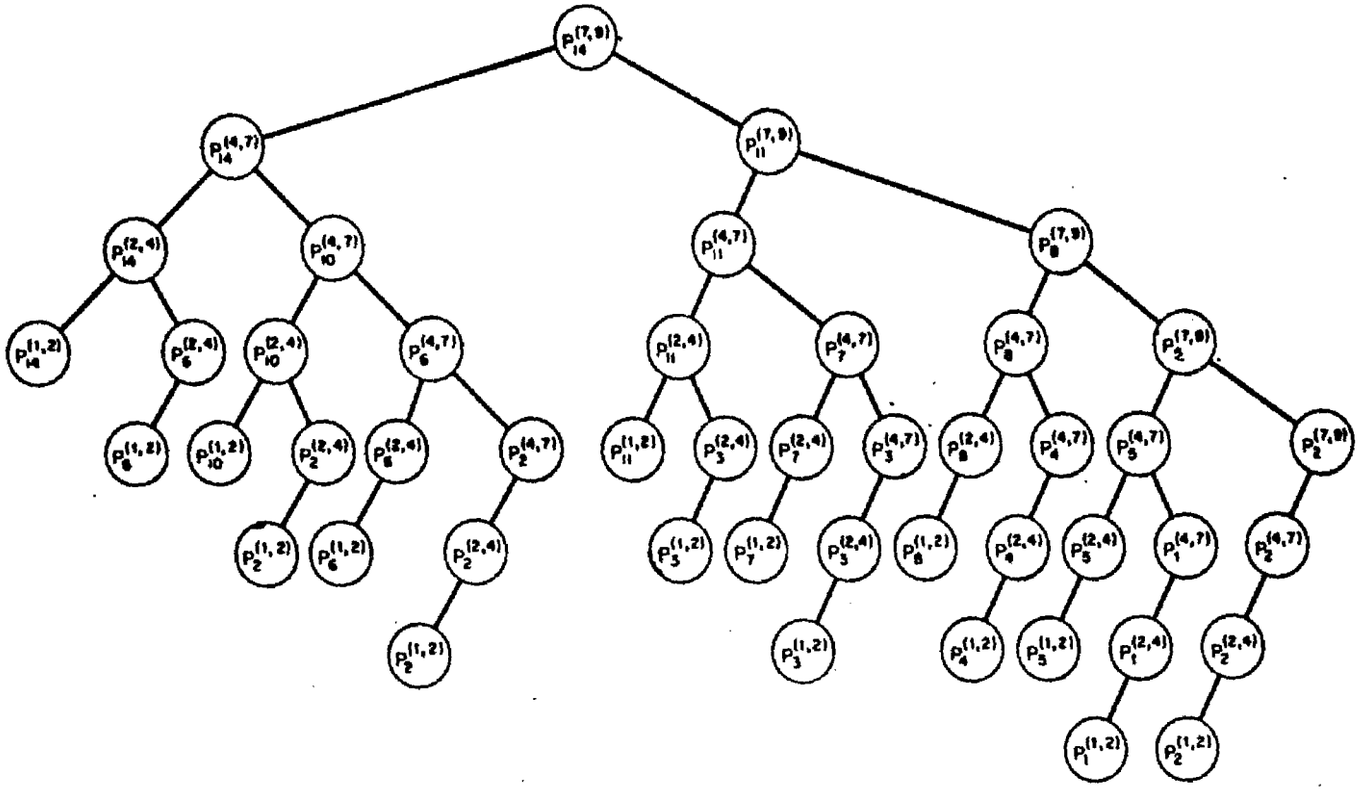

Figure 2.10. Binary tree representation of the GCAA

$P_{14}^{(7,9)}$ and traverse the left subtree until $P_{14}^{(1,2)}$ is reached. Compute the value of $P_{14}^{(1,2)}$, go up to its root $P_{14}^{(2,4)}$, and traverse its right subtree until the terminal node where the value of $P_6^{(1,2)}$ is computed. As $P_6^{(2,4)}$ has no right subtree, the value of $P_6^{(2,4)}$ equals $P_6^{(1,2)}$ plus $d_{2,4}$. Now this value and $P_{14}^{(1,2)}$ can be used to compute $P_{14}^{(2,4)}$ as max $(P_{14}^{(1,2)}, P_6^{(2,4)}) + d_{2,4}$ as presented in Algorithm GCAA. When the root node $P_{14}^{(7,9)}$ is computed, the calculation is complete. What was referred to as the maximum depth of the GCAA is actually the maximum depth of the above tree. This is the number of packets on the longest path of the tree; in our example $P_{14}^{(7,9)}$ to $P_2^{(1,2)}$ which is eight.

Chapter 2 has presented a novel algorithm for the computation of total service time in a packet switched store-and-forward communication network. Faster results than using a simulation are achieved due to selective calculation, parallel computation and no duplication of effort. The recursive algorithm simplifies the task of the analyst as only two parameters change for each design to be tested: the number of packets and the destination node. The logic and structure of the program is the same. A simulation program, however, would require the structure to be changed for each test design--a burdensome and time-consuming task for the analyst. The graphic representation can be constructed quickly and allows for a further speed-up in computation. As in the case of a two-arc three-node network, the fact that

the Independence Assumption, with its homogenizing effect on message flows, is not required to carry out the analysis for the computation of exact service time means that a true-to-life result is obtained. A unique and powerful new tool is thus available to aid in the complex task of network design.

2.3.4 Extension to a General Network

Given a general network as in Figure 2.11, the GCAA can be applied to each path from source, node 1, to destination, node 9. Assuming that no node is visited twice in a path, the path will contain exactly N nodes and N-1 arcs. Under these circumstances the GCAA will provide the exact total service time for each path given a number of packets to be shipped.

An analysis of these paths will supply the network designer with the best routing configuration. The network might contain one optimal path over which all packets should be sent, regardless of their number. Or, the optimal path might be a function of the number of packets at the source node. For example, one path might be the fastest for one to thirty packets, another path might be fastest for thirty to one hundred packets, and a third for more than one hundred. This would entail an analysis of the paths under varying traffic loads. If, however, an average number of packets is expected at the source node, then the best route for this number could be used. If the average increases or decreases

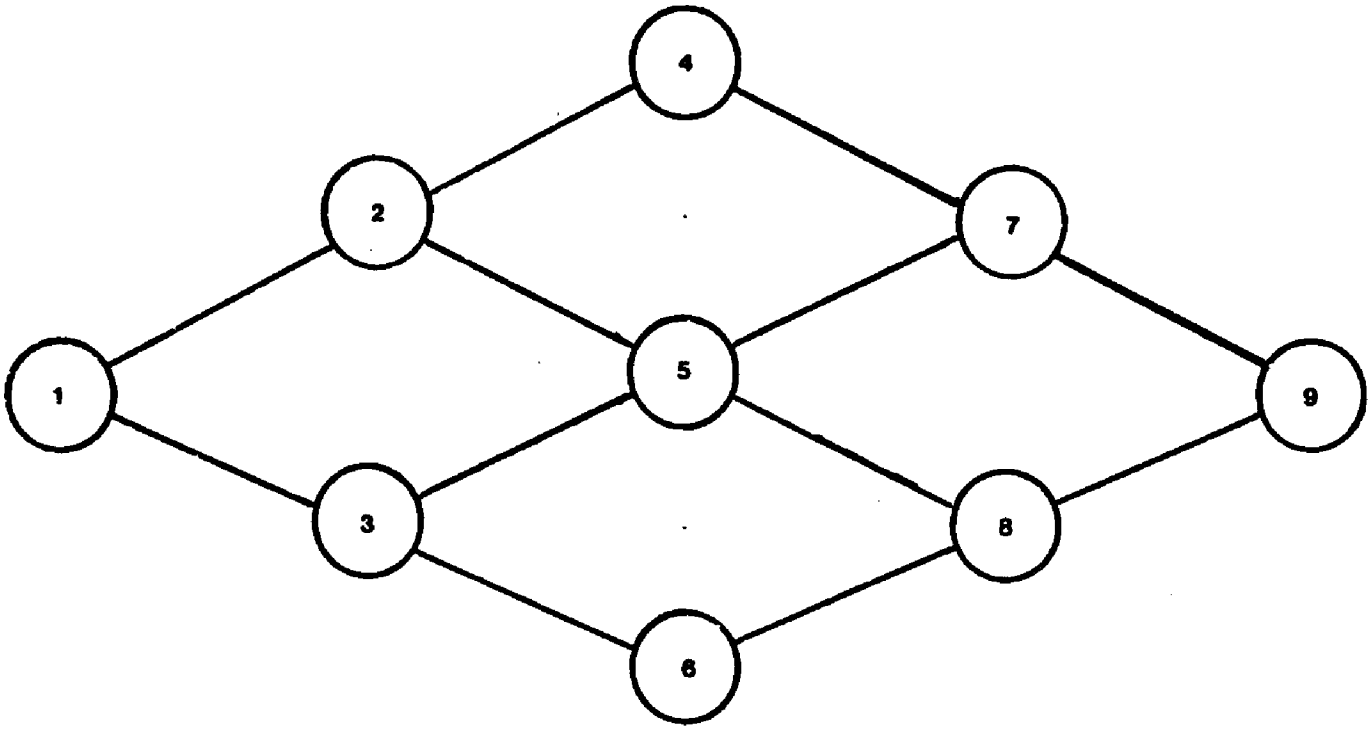

Figure 2.11. General directed network labeled with distance and capacity for each arc

beyond a certain range, then other paths would be selected to correct for the new traffic load.

A third possibility for routing is to select alternate routes once an optimal path has been decided upon. If, for example, path (1,2,4,7,9) was the optimum, it might be possible to further speed up the flow of packets by shipping in parallel a portion of the packets over path (1,3,6,8,9). This pipelining effect is one of the main advantages of packet-switching. An alternate path that intersects the primary route and has an arc in common with it, e.g., secondary path (1,3,5,7,9), might void the analysis of the GCAA performed on the primary route. Either the number of packets arriving at node 7 might be more than accounted for by the algorithm, or the timing might be off, i.e., the packets from the alternate route may arrive later than the GCAA expected them. The analysis of this situation remains a topic for further research.

CHAPTER 3

THE BOUNDED SOLUTION: BOUNDING TECHNIQUES FOR TOTAL SERVICE TIME IN A PACKET-SWITCHED NETWORK

3.0 Introduction

The previous chapter dealt with finding an exact value for total service time in a packet-switched network. As crucial a tool as it is for network design, finding an exact solution is still a time-consuming process to apply to all possible configurations during the design stage. A more practical approach to take during this stage is to eliminate designs from consideration before computing exact service time via the GCAA. Of course, this is a viable possibility only if the elimination process is faster than the exact solution algorithm.

This chapter will present upper and lower bounding techniques to help eliminate those designs that do not fall into the desired service time range of the initial user specifications. In addition, a handle is obtained on those networks that do satisfy the service time criterion because, though the exact value is not known, bounds are known even before the GCAA is used. Thus, even among the "passing" designs a decision can be made as to which order to apply them to the GCAA based upon the spread of their bounds. The tighter the spread, the better the idea of what the exact

value is and how useful that design is to the analyst. Those designs that show more promising results can be tested first to further speed up the design analysis.

3.1 Two-Arc Three-Node Network (Rudowsky & Rootenberg, 1978)

Consider the two-arc three-node network in Figure 3.1. To ship one packet from node 1 to node 3 requires eight time units, $t_{12}+t_{23}$. Three packets would require the same time for this trip. In both cases there is no queueing time incurred at node 2. As the capacity of arc (1,2) is three units, shipping six packets would require that they be sent as two blocks of three packets each. The first block of three packets arrives at node 2 at time two, assuming it started out from node 1 at time zero, and at node 3 at time eight. The second block arrives at node 2 at time four, as it cannot use arc (1,2) before time two since the first block is using it, and reaches node 3 after six more time units, i.e., time ten. When the second block reaches node 2, arc (1,2) is not used to capacity so again there is no queueing. However, to ship nine packets would cause queueing to occur. Specifically, at time six there would be six packets on arc (2,3), blocks 1 and 2, and three packets just arriving at node 2 from node 1. Since the capacity of arc (2,3) is eight units, only two additional packets from the third block can be shipped over arc (2,3) at time six. This leaves one packet from block 3 queued at node 2 until time eight when block 1 reaches node 3 and relinquishes its use of arc (2,3). A total of fourteen time units is thus required to ship nine packets from node 1

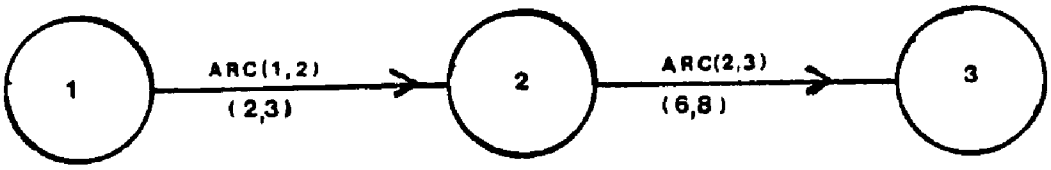

Figure 3.1, Two-arc three-node directed network

to node 3. If there would be no queueing at node 2 (it will be explained later under what circumstances this can occur) only twelve time units would be required for the flow to complete its traversal. This value can be arrived at by calculating the number of blocks required to ship the packets to node 2, multiplying by $t_{1,2}$ and adding $t_{2,3}$ to the result. When there is no queueing, all that is needed to compute total service time is the arrival time of the last block of packets to node 2 and then to add on the time to cross arc (2,3), $t_{2,3}$. No repetitive algorithm or simulation is required. What is sought is a method to eliminate queueing and achieve an acceptable estimate on the service time.

3.1.1 Queueless Networks

The amount of traffic on an arc and the relationship between the parameters of adjacent arcs dictate whether queueing will occur or not. For example, if the capacities of arc (1,2) and arc (2,3) are equal and the distances are equal, case (E,E), then no queueing will ever occur no matter how many packets are sent over the path. At the same time that block 2 reaches node 2, block 1 reaches node 3, because $t_{1,2}$ equals $t_{1,3}$. As long as arc (2,3) is free as soon as or before it is needed no queueing will occur at node 2. Thus, when capacities of adjacent arcs are equal and $d_{1,2}$ is greater than $d_{2,3}$, i.e., case (G,E), there will also never be any queueing as arc (2,3) will become free even before the next block arrives at node 2. In case (E,E), arc (2,3) becomes free exactly when the next packet arrives. Two other cases also do

not incur queueing, they are case (E,L) - $d_{1,2}=d_{2,3}$, $C_{1,2}<C_{2,3}$ and case (G,L) - $d_{1,2}>d_{2,3}$, $C_{1,2}<C_{2,3}$. The fact that these four cases do not involve the calculation of queueing time enables us to compute a reasonable upper and lower bound on the total service time in a two-arc three-node network.

First a proof will be stated showing that in the four cases--(E,E), (E,L), (G,E) and (G,L)--queueing never occurs at node 2 during the packet flow. Then the upper and lower bound calculations will be stated, analyzed and illustrated by example.

In order to prove the lack of queueing in the above four cases, the analysis presented in section 2.1 will be employed. The basic idea is to treat the second arc of a two-arc three-node network as a group of parallel, independent sub-arcs each with a capacity of one unit and distance equal to the original distance. By cyclically allocating the packets from node 2, along these sub-arcs, to node 3 and comparing the arrival time at node 3 of a packet using a sub-arc and the arrival time at node 2 of the next packet to use that same sub-arc, it can easily be determined whether queueing occurs or not. For example, view the capacity of arc (2,3) in Figure 3.1, eight units, as meaning that there are eight parallel sub-arcs from node 2 to node 3 each with a capacity of one unit and distance of six units. The cyclic allocation algorithm assigns packet 1 to sub-arc 1, packet 2 to sub-arc 2, . . . packet 8 to sub-arc 8, packet 9 to sub-arc 1, etc. Thus,

packet 1, which uses sub-arc 1, arrives at node 2 at time two and at node 3 at time eight. Packet 9, the next to use sub-arc 1, arrives at time six at node 2. It cannot be shipped over sub-arc 1 because packet 1 has not arrived at node 3 by time six. Only at time eight does sub-arc 1 become free, allowing packet 9 to be forwarded; thus packet 9 queues for two time units. If, for example, $d_{2,3}$ was three units instead of six, then packet 1 would arrive at node 3 at time five rather than time eight. In this instance there would be no queueing time incurred for packet 9 because sub-arc 1 would be free when it arrives at node 2, i.e., at time six. This simple comparison of the arrival time of a packet at node 3 and the arrival time at node 2 of the next packet to use the same sub-arc indicates whether queueing will occur or not.

Given a two-arc three-node network as in Figure 3.2 with associated capacities and distances, show that for:

Case (E,E): $d_{1,2} = d_{2,3}$ and $C_{1,2} = C_{2,3}$

Case (E,L): $d_{1,2} = d_{2,3}$ and $C_{1,2} < C_{2,3}$

Case (G,E): $d_{1,2} > d_{2,3}$ and $C_{1,2} = C_{2,3}$

Case (G,L): $d_{1,2} > d_{2,3}$ and $C_{1,2} < C_{2,3}$

no queueing occurs in the network for any number of packets shipped. Let T represent the total number of packets.

The method of proof will be to show that for any two packets consecutively using the same sub-arc of arc (2,3), the completion time of the former is less than or equal to the arrival time at node 2 of the latter. Completion time refers to the arrival time of a packet at the destination node, in this case node 3.

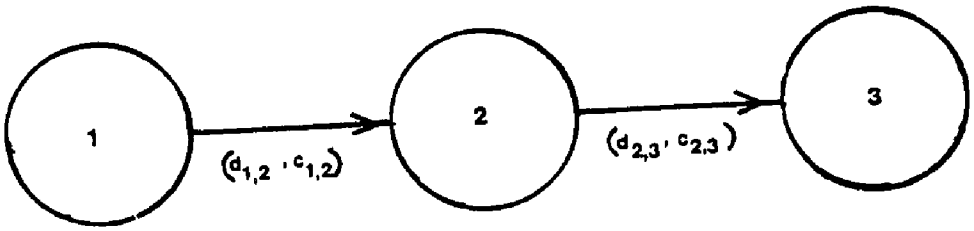

Figure 3.2. General two-arc three-node directed network

As shown by the analysis in section 2.1, the network in Figure 3.2 can be analyzed by Table 3.1. If T packets are to be shipped from node 1 to node 3, the packets that use each sub-arc (the columns) can be listed as in Table 3.1. There are $C_{2,3}$ columns as there are $C_{2,3}$ sub-arcs, one for each unit of capacity of arc (2,3). First it will be shown that for any packet in the first block of packets, the packet from the next block that uses the same sub-arc arrives no earlier at node 2 than when the sub-arc becomes free, thus incurring no queueing. Then it will be shown for any two packets, $k \cdot C_{2,3}+j$ and $(k+1) \cdot C_{2,3}+j$ (both using the same sub-arc), the completion time of the former is no later than the arrival time of the latter at node 2, so again no queueing occurs. Thus, by the Induction Hypothesis it is true for any two packets consecutively using the same sub-arc. The completion time of the $k \cdot C_{2,3}+j^{th}$ packet is

$$\left\lceil \frac{k \cdot C_{2,3}+j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3}$$

$$1 \leq j < C_{2,3}, \quad 0 \leq k \leq \left\lfloor \frac{T}{C_{2,3}} \right\rfloor$$

where $\lceil a \rceil$ is the least integer upper bound of a and $\lfloor a \rfloor$ is the greatest integer lower bound of a.

The arrival time at node 2 of the $(k+1) \cdot C_{2,3}+j^{th}$ packet is

$$\left\lceil \frac{(k+1) \cdot C_{2,3}+j}{C_{1,2}} \right\rceil \cdot t_{1,2}.$$

Table 3.1

Flow of Individual Packets on Sub-Arcs of Arc (2,3)

Sub-Arc	1	2	...	j	...	REM $\frac{T}{C_{2,3}}$ *	...	$C_{2,3}$
Block 0	1	2	...	j	...	REM $\frac{T}{C_{2,3}}$	...	$C_{2,3}$
1	$C_{2,3}+1$	$C_{2,3}+2$	...	$C_{2,3}+j$	...	$C_{2,3}+$ REM $\frac{T}{C_{2,3}}$	...	$2 \cdot C_{2,3}$
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
k	$k \cdot C_{2,3}+1$	$k \cdot C_{2,3}+2$	...	$k \cdot C_{2,3}+j$	...	$k \cdot C_{2,3}+$ REM $\frac{T}{C_{2,3}}$	...	$(k+1) \cdot C_{2,3}$
k+1	$(k+1) \cdot C_{2,3}+1$	$(k+1) \cdot C_{2,3}+2$	...	$(k+1) \cdot C_{2,3}+j$	...	$(k+1) \cdot C_{2,3}+$ REM $\frac{T}{C_{2,3}}$	...	$(k+2) \cdot C_{2,3}$
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
$\left\lfloor \frac{T}{C_{2,3}} \right\rfloor$ **	$\left\lfloor \frac{T}{C_{2,3}} \right\rfloor \cdot C_{3,3}+1$	$\left\lfloor \frac{T}{C_{2,3}} \right\rfloor \cdot C_{2,3}+2$	...	$\left\lfloor \frac{T}{C_{2,3}} \right\rfloor \cdot C_{2,3}+j$	...	T		

*REM $\frac{x}{z}$ is the remainder of the division of x by z.

**$\lfloor x \rfloor$ is the largest integer less than or equal to x.

The goal is to prove that

$$\left\lceil \frac{k \cdot C_{2,3} + j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil \frac{(k+1) \cdot C_{2,3} + j}{C_{1,2}} \right\rceil \cdot t_{1,2}$$

(Equation 3.1)

for any $k \in \left\{0 \leq k \leq \left\lfloor \frac{T}{C_{2,3}} \right\rfloor\right\}$, $j \in \{1 \leq j \leq C_{2,3}\}$ and

j,k are integer.

This will be done separately for each group of cases having similar distance values.

Cases (E,E) and (E,L):

$d_{1,2} = d_{2,3}$, $t_{1,2} = t_{2,3}$ and $C_{1,2} \leq C_{2,3}$

Let $C_{2,3} = w \cdot C_{1,2}$, $w \geq 1$.

For k=0 (i.e., the first group of $C_{2,3}$ packets to use arc (2,3)) we have from Equation 3.1:

$$\left\lceil \frac{j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil \frac{C_{2,3} + j}{C_{1,2}} \right\rceil \cdot t_{1,2}$$

Let $\frac{j}{C_{1,2}} = v$ and replace $t_{2,3}$ with $t_{1,2}$ as they are equal.

$$(\lceil v \rceil + 1) \cdot t_{1,2} \leq \lceil w+v \rceil \cdot t_{1,2}$$

(Note that $\lceil a + 1 \rceil = \lceil a \rceil + 1$ for all a.)

$$\lceil v+1 \rceil \cdot t_{1,2} \leq \lceil v + w \rceil \cdot t_{1,2}$$

The last equation is true since $w \geq 1$.

Assume Equation 3.1 is true for k=1,2, . . ., n-1.

For k = n we have

$$\left\lceil \frac{n \cdot C_{2,3} + j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil \frac{(n+1) C_{2,3} + j}{C_{1,2}} \right\rceil \cdot t_{1,2}$$

$$\left(\left\lceil n\cdot w+\frac{j}{C_{1,2}}\right\rceil + 1\right)\cdot t_{1,2} \leq \left\lceil (n+1)\cdot w + \frac{j}{C_{1,2}}\right\rceil \cdot t_{1,2}$$

Let $\frac{j}{C_{1,2}} = v$.

$$(\lceil n\cdot w+v\rceil + 1)\cdot t_{1,2} \leq \lceil n\cdot w+v+w\rceil \cdot t_{1,2}$$

$$(\lceil n\cdot w+v+1\rceil) \cdot t_{1,2} \leq \lceil n\cdot w+v+w\rceil \cdot t_{1,2}$$

Again the above equation is valid since $w \geq 1$.

Thus, by the Induction Hypothesis, Equation 3.1 is true for all k, i.e., for all packets shipped, and no queueing occurs.

Cases (G,E) and (G,L):

$d_{1,2} > d_{2,3}$, $t_{1,2} > t_{2,3}$ and $C_{1,2} \leq C_{2,3}$

Let $t_{1,2} = w\cdot t_{2,3}$, $w>1$ and $C_{2,3} = x\cdot C_{1,2}$, $x \geq 1$.

For k=0 we have from Equation 3.1:

$$\left\lceil \frac{j}{C_{1,2}}\right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil \frac{C_{2,3}+j}{C_{1,2}}\right\rceil \cdot t_{1,2}$$

Let $\frac{j}{C_{1,2}} = v$.

$$\lceil v\rceil \cdot t_{1,2} + t_{2,3} \leq \lceil x + v\rceil \cdot t_{1,2}$$

Since x is at least 1, substitute 1 for x into the right-hand side.

$$\lceil v\rceil \cdot t_{1,2} + t_{2,3} \leq \lceil v + 1\rceil \cdot t_{1,2}$$

Note: $\lceil q + 1\rceil \cdot z = \lceil q\rceil \cdot z + z$ for all q and z.

$\lceil v\rceil \cdot t_{1,2} + t_{2,3} \leq \lceil v\rceil \cdot t_{1,2} + t_{1,2}$ which is true since $t_{1,2} > t_{2,3}$.

Assume true for k = 1,2 . . . n-1.

For k = n we have

$$\left\lceil \frac{n \cdot C_{2,3}+j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil \frac{(n+1) \cdot C_{2,3}+j}{C_{1,2}} \right\rceil \cdot t_{1,2}$$

$$\left\lceil n \cdot x + \frac{j}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3} \leq \left\lceil (n+1) \cdot x + \frac{j}{C_{1,2}} \right\rceil \cdot t_{1,2}$$

Let $\frac{j}{C_{1,2}} = v$

$$\lceil n \cdot x+v \rceil \cdot t_{1,2} + t_{2,3} \leq \lceil (n+1) \cdot x+v \rceil \cdot t_{1,2}$$

Since x is at least 1, substitute 1 for x in the above equation.

$$\lceil n+v \rceil \cdot t_{1,2} + t_{2,3} \leq \lceil n+v+1 \rceil \cdot t_{1,2}$$

$$\lceil n+v \rceil \cdot t_{1,2} + t_{2,3} \leq \lceil n+v \rceil \cdot t_{1,2} + t_{1,2}$$

The last equation is true since $t_{1,2} > t_{2,3}$. Thus by the Induction Hypothesis, Equation 3.1 holds for cases (G,E) and (G,L) and no queueing occurs over such a network.

Having shown that in the four cases (E,E), (E,L), (G,E) and (G,L) no queueing can occur during the packet flow, upper and lower bounds to the total service time for the remaining five cases, i.e., (E,G), (G,G), (L,L), (L,E) and (L,G), are sought. In those five cases it can be shown that queueing occurs under most conditions. Under some restricted circumstances, queueing will not be incurred in some of those five cases; restricted enough so as not to be useful for purposes of an upper bound calculation.

3.1.2 Upper Bound Scheme

The scheme for computing an upper bound on the total service of T packets flowing from node 1 to node 3 in Figure 3.2 is as follows:

(1) set $d_{1,2} = \max(d_{1,2}, d_{2,3})$, $t_{1,2} = \max(t_{1,2}, t_{2,3})$

$C_{1,2} = \min(C_{1,2}, C_{2,3})$

(2) Total Service Time = $\left\lceil \frac{T}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3}$.

The first step converts the given network into one of the four cases known not to incur queueing. By setting $d_{1,2}$ to the maximum of $d_{1,2}$ and $d_{2,3}$, $d_{1,2}$ is now greater than or equal to $d_{2,3}$. On the other hand, setting $C_{1,2}$ to the minimum of $C_{1,2}$ and $C_{2,3}$ means $C_{1,2}$ is less than or equal to $C_{2,3}$. To summarize, we have $d_{1,2} \geq d_{2,3}$ and $C_{1,2} \leq C_{2,3}$ which is either (E,E), (E,L), (G,E), or (G,L). Once it is known that no queueing occurs, because of step (1), the total service time can be computed by finding the arrival time of the last block of packets at node 2, i.e., computing $\left\lceil \frac{T}{C_{1,2}} \right\rceil \cdot t_{1,2}$.

Since we know that there is no queue at node 2 at any time, the arrival time of the last block of packets at node 3 is calculated by adding $t_{2,3}$ to the last arrival time at node 2. Thus, we have step (2). For any of the cases (E,E), (E,L), (G,E), or (G,L), when applied to the above upper bound algorithm, the exact value of the total service time will be computed. Approximations will be calculated for the remaining five cases. For example, the network with $d_{1,2} = 5$, $d_{2,3} = 6$, $C_{1,2} = 8$ and $C_{2,3} = 9$ is case (L,L). To ship 50 packets, the

exact solution from the cyclic allocation algorithm yields a value of 45 time units for the total service time. Applying the upper bound scheme yields:

(1) $d_{1,2} = 6$, $t_{1,2} = 6$ and $C_{1,2} = 8$

(2) Total Service Time = $\left\lceil \frac{50}{8} \right\rceil \cdot 6 + 6 = 48$ units.

For the example

$d_{1,2} = 5$, $d_{2,3} = 10$, $C_{1,2} = 8$, and $C_{2,3} = 13$, with

$T = 50$ messages

we find the exact solution to be 50 time units while the upper bound approximation is 80. It seems that the closer the values of $d_{1,2}$ and $C_{1,2}$ are to their original values, $d_{1,2}$ and $C_{1,2}$, the better the approximation.

The logic behind the algorithm is that by selecting the maximum value of $d_{1,2}$ and $d_{2,3}$ for the value of $d_{1,2}$, the shipping time increases because the time to traverse arc (1,2) has grown. Similarly, by choosing the minimum between $C_{1,2}$ and $C_{2,3}$ for $C_{1,2}$, the increase in the number of blocks of packets delays the departure time of the last block from node 1 and thus its arrival time at node 2. It can be seen that step (1) of the procedure increases (or at best leaves as is) the total service time by selecting the maximum of the distances and minimum of the capacities. Thus, an upper bound is computed on the total service time.

3.1.3 Lower Bound Scheme

The lower bound procedure reverses the process of the upper bound technique. Here the goal is to increase the

number of packets that can travel over the network and decrease the length of the arcs while still avoiding queueing at node 2. These two changes will speed the flow of packets, thereby decreasing the total service time and thus provide a lower bound.

(1) Set

$$d_{2,3} = \min (d_{1,2}, d_{2,3})$$

$$t_{2,3} = \min (t_{1,2}, t_{2,3})$$

$$C_{2,3} = \max (C_{1,2}, C_{2,3})$$

(2) Total Service time = $\left\lceil \frac{T}{C_{1,2}} \right\rceil \cdot t_{1,2} + t_{2,3}$.

Again, the first step converts the network into a non-queueing case as $d_{1,2}$ is greater than or equal to $d_{2,3}$ and $C_{1,2}$ is less than or equal to $C_{2,3}$. Based upon the new values of $t_{2,3}$ and $C_{2,3}$, the total service time can be computed with the simple formula in step (2).

Applying this procedure to the examples used in section 3.1.2 yields the following results. The network with $d_{1,2} = 5$, $d_{2,3} = 6$, $C_{1,2} = 8$ and $C_{2,3} = 9$ is changed so that the new value of $d_{2,3}$ is 5. This yields a value for total service time of 40 units. The second example, with $d_{1,2} = 5$, $d_{2,3} = 10$, $C_{1,2} = 8$ and $C_{2,3} = 13$, has a lower bound of 40 units.

Summarizing our results we find that for a fifty packet message the first network has an upper bound of 48, a lower bound of 40 with an exact solution of 45. The second example has an upper bound of 80 and a lower bound of 40 with 50 being the exact solution. Thus, the above procedures for

an upper and lower bound on total service time provide the analyst with a range to test the initial design specifications for this parameter. Given a desired value for the total service time, one can quickly test a two-arc three-node network to see if the required service time can be provided under varying number of packets. If the value falls outside of the bounds, then the network can be dropped from further study; if it falls within the bounds, the network can be ranked for further analysis with the GCAA. For example, a network with close upper and lower bounds within which falls the desired service time value might be ranked higher for further study than a network with bounds further apart. One can be more definite that the network specification will be met. The time required to compute the bounds is very small in comparison to the time complexity of the CAA--only a simple comparison and one-line formula are needed. Thus, the time spent in computing bounds is well worth it because much more time would be spent if only the CAA were to be applied to each network. Bounding and the subsequent elimination of infeasible networks from further study can thus lead to a savings in time and money during the analysis and design stages of network building.

3.2 N-1 Arc N-Node Network

The previous section detailed and analyzed novel techniques for calculating upper and lower bounds on the total service time in a two-arc three-node network. To further improve their usefulness to a network analyst, these bounding

schemes will be extended first to networks of N-1 arcs and N nodes and then to the general network case (any number of nodes and arcs). This will allow the analyst to reduce the number of designs to be studied by eliminating those that do not meet the required operating characteristic of total service time. This will in turn reduce the time and cost of the design process and provide the analyst with an optimal or near optimal configuration.

3.2.1 Upper Bound Scheme

In order to extend the upper bound scheme for networks with two arcs and three nodes (section 3.1.2) to networks with N-1 arcs and N nodes, the latter must be viewed as a series of overlapping two-arc three-node paths. For example, in the network of Figure 3.3 there are three paths of two arcs and three nodes--they are paths (4,7,9), (2,4,7) and (1,2,4). The paths have purposefully been listed with the destination node, node 9, appearing in the first path of the list and the source node, node 1, appearing last because this order must be followed when applying the upper bound scheme. This means that the path with the destination node is transformed first, the one prior to it second, and the path with the source node is transformed last.

For the network to be non-queueing, the distance parameter of each arc, starting from the source node, must be no smaller than the one following it. This is insured by starting the upper bound scheme from the destination node as then the distance of the arc to be changed is equal to the

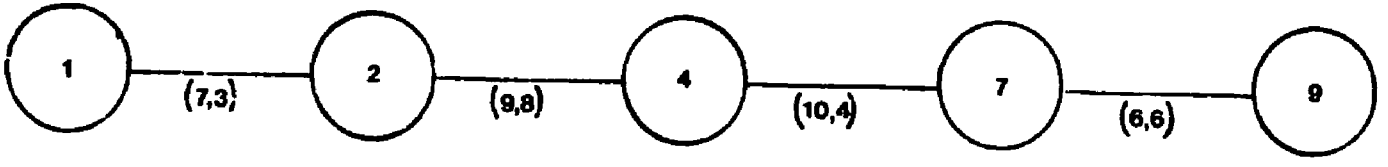

Figure 3.3. N-1-arc N-node directed network labeled with distance and capacity for each arc

maximum of its distance and the distance of the arc that follows. But the distance of this latter arc has already been set to the maximum of its own distance and the distance of the arc that follows it. Thus, adjacent arc distances are non-decreasing beginning with the source node and moving to the destination node. In path (4,7,9) of Figure 3.3 the distance of arc (4,7) is unchanged since it is larger than $d_{7,9}$. However, $d_{2,4}$ is changed to ten which is the maximum of $d_{2,4}$ and $d_{4,7}$. But $d_{4,7}$ is already the maximum of $d_{4,7}$ and $d_{7,9}$; thus, in effect, $d_{2,4}$ is the maximum of $d_{2,4}$, $d_{4,7}$ and $d_{7,9}$. Similarly, it can be shown that $d_{1,2}$ is the maximum of $d_{1,2}$, $d_{2,4}$, $d_{4,7}$ and $d_{7,9}$. In general $d_{i_k,i_{k+1}}$ = maximum $(d_{i_k,i_{k+1}}, d_{i_{k+1},i_{k+2}}, \ldots d_{i_{N-1},i_N})$ where k is less than or equal to N-1.

The reverse is true for capacity. Since the minimum of the two capacities is chosen, the result is that the capacity of one arc is no larger than the capacity of the following arc. The capacity of the first arc in the path will be the smallest capacity in the path and will be used to determine the number of blocks of packets into which the message will be divided. In this case

$C_{i_k,i_{k+1}}$ = minimum $(C_{i_k,i_{k+1}}, C_{i_{k+1},i_{k+2}}, \ldots C_{i_{N-1},i_N})$

where k is less than or equal to N-1.

Given the N-1 arc N node network in Figure 3.3, begin by converting the two-arc three-node path with node 9 as its destination, i.e., path (4,7,9), into a non-queueing case--either (E,E), (E,L), (G,E) or (G,L). By employing the upper bound technique we find that no change need be made in this

path as it already is a non-queueing path, i.e., case (G,L). Next, the arc containing node 9 is dropped from further consideration and the previous intermediate node, node 7, is the new destination. The two-arc three-node path (2,4,7) is also converted to a non-queueing case. In this instance the capacity of arc (2,4) is decreased to four and the distance is increased to ten so that path (2,4,7) is now case (E,E). Continuing this process, path (1,2,4) is next examined and $d_{1,2}$ is increased to ten while $C_{1,2}$ remains three so that the path is case (E,L). Since there is no two-arc three-node path with node 2 as the destination node, the conversion process ends. Figure 3.4 depicts the transformation of Figure 3.3 into a non-queueing network to solve for upper bounds.

Having completed the transformation phase, begin the computation phase. Given a number of packets at node 1, ship blocks of maximum size of the capacity of arc (1,2), in this case three. No queueing will occur in the path because it has been specifically converted to be queueless. For example, to ship sixteen packets from node 1 to node 9, block the packets into size three. In total there are six blocks, five of size three and one of size one. They depart node 1 starting at time zero and every seven time units thereafter, as $t_{1,2}$ equals seven. The last block leaves node 1 at time fifty and arrives at node 9 at time $50 + t_{1,2} + t_{2,4} + t_{4,7} + t_{7,9}$ or time eighty-six.

All that is needed to compute the upper bound of the

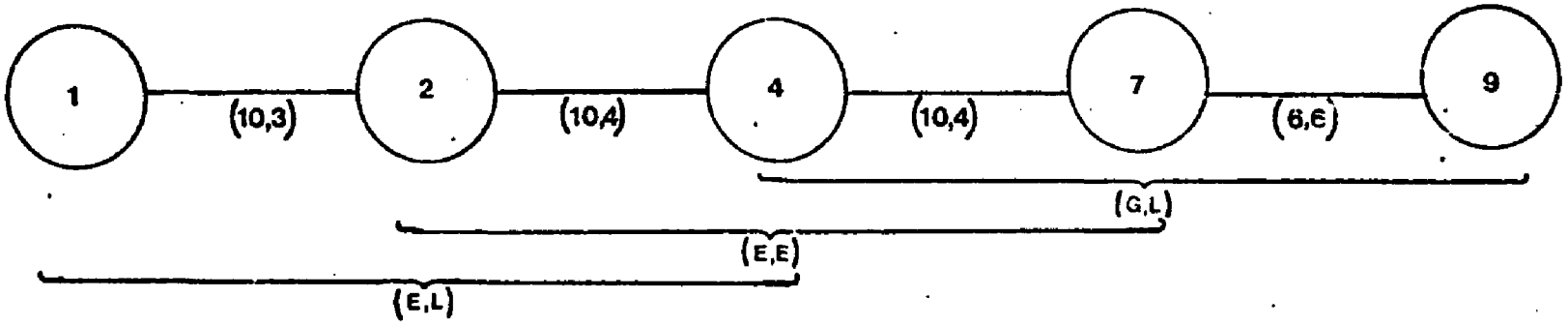

Figure 3.4. Transformation of Figure 3.3 into a non-queueing network to solve for upper bounds

total service time of T packets over path P is the departure time of the last block from the source node and the sum of $t_{i,j}$ for all arcs in the path. Since no queueing will occur, service time is equal to shipping time. The former quantity can be computed based upon the analysis of section 2.1 as follows: Let C_{i_1,i_2} equal the capacity of the first arc, by the nature of the upper bound procedure it will always be the smallest capacity in the path, and let t_{i_1,i_2} equal the time to traverse the first arc. The departure time of the last block of packets from node 1 is:

$$\left(\left\lceil \frac{T}{C_{i_1,i_2}} \right\rceil - 1\right) \cdot t_{i_1,i_2}$$

The arrival time at the destination node is computed by adding to the above time the sum of traversal time for each arc in the path, i.e., $\sum t_{i_m,i_n}$ where $(i_m,i_n) \in P$. Thus the complete equation for an upper bound is

$$\left(\left\lceil \frac{T}{C_{i_1,i_2}} \right\rceil - 1\right) \cdot t_{i_1,i_2} + \sum t_{i_m,i_n} \text{ where } (i_m,i_n) \in P. \qquad \text{(Equation 3.2)}$$

Summary for calculating an upper bound on an N-1-arc N-node network as illustrated in Figure 3.5:

Phase I - Transformation

(1) Set k = N (start from the destination node), if k < 3 STOP (not a two-arc three-node path)

(2) Set $t_{i_{k-2},i_{k-1}}$ = maximum $(t_{i_{k-2},i_{k-1}}, t_{i_{k-1},i_k})$

$C_{i_{k-2},i_{k-1}}$ = minimum $(C_{i_{k-2},i_{k-1}}, C_{i_{k-1},i_k})$

(3) Set k = k-1 (move towards the source node)

if k-2 < 1 go to Phase II (no more two-arc three-node paths)

(4) Go to step (2)

Figure 3.5. General N-1-arc N-node directed network

Phase II - Computation

(1) Upper bound =

$$\left(\left\lceil \frac{T}{C_{i_1,i_2}} \right\rceil - 1 \right) \cdot t_{i_1,i_2} + \sum t_{i_m,i_n}$$

where i_m and i_n are adjacent nodes in the path.

3.2.2 Lower Bound Scheme

As was the case with the upper bound scheme, the lower bound scheme of section 3.1.3 can also be extended to a network of N-1 arcs and N nodes. Again, the network of Figure 3.3 must be viewed as a series of overlapping two-arc three-node paths. The paths are (1,2,4), (2,4,7) and (4,7,9)--here they are ordered starting from the source, node 1, rather than from the destination, node 9, as was the case in the upper bound scheme.

In order to transform Figure 3.3 into a non-queueing network to solve for a lower bound, the distance of an arc cannot be less than the distance of the arc that follows it. Since the lower bound scheme selects the minimum of the distances as the new distance for the second arc in the two-arc three-node path, the process must begin at the source node to satisfy the non-increasing rule of adjacent arc distance parameters. Thus, $d_{2,4}$ = minimum $(d_{1,2}, d_{2,4})$ and $d_{4,7}$ = minimum $(d_{2,4}, d_{4,7})$ = minimum $(d_{1,2}, d_{2,4}, d_{4,7})$--or in general $d_{i_k,i_{k+1}}$ = minimum $(d_{i_1,i_2}, d_{i_2,i_3}, \ldots, d_{i_k,i_{k+1}})$ where k is less than or equal to N-1. The same order of operation, from source to destination, will have the opposite

effect on capacity since the maximum of the two adjacent arcs becomes the capacity for the second arc of the two-arc three-node network. Here $C_{i_k,i_{k+1}}$ = maximum (C_{i_1,i_2}, C_{i_2,i_3} . . ., $C_{i_k,i_{k+1}}$) where k is less than or equal to N-1. The result will be that the first arc will have the smallest capacity. This fact will be used to determine the number of packets in a block.

To compute a lower bound for the network of Figure 3.3, begin by applying the lower bound procedure for two-arc three-node networks to path (1,2,4). The distance of arc (2,4) is decreased to seven while the capacity remains the same to make path (1,2,4) the non-queueing case (E,L). Dropping node 1 and examining path (2,4,7), one sees that $C_{4,7}$ has to be increased to eight and $d_{4,7}$ reduced to seven to arrive at case (E,E) for the path. In the next two-arc three-node path, path (4,7,9), the capacity of arc (7,9) is increased to eight, transforming the path into case (G,E). As there are no further paths with two arcs and three nodes, the transformation phase stops. See Figure 3.6 for the new configuration after applying the lower bound procedure in the above manner.

The computation for total service time follows the same formula as shown for the upper bound, i.e., equation 3,2. Thus, with sixteen packets at node 1 in Figure 3,6, equation 3.2 yields a lower bound on the total service time of sixty-five. The exact value of total service time is sixty-nine. In this example the lower bound of sixty-five was closer to

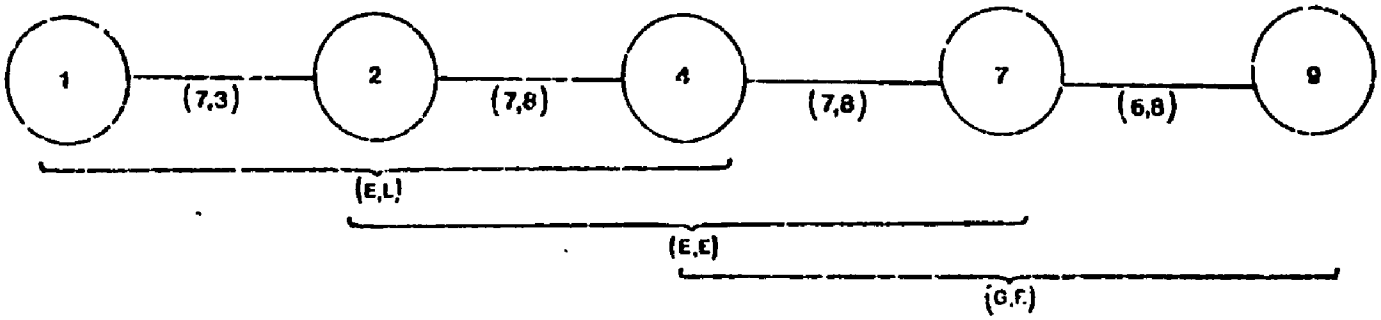

Figure 3.6. Transformation of Figure 3.3 into a non-queueing network to solve for lower bounds

the true solution though this is known only via the GCAA, not through the bounding procedures.

The network designer can derive from this analysis that if a message of sixteen packets (actually eighteen is the maximum since the last block can contain up to three packets) or less arrives at node 1 every eighty-six time units, it will take between sixty-five and eighty-six time units to arrive at the destination, node 9. This is useful in that it tells the analyst that if a maximum total service time of fifty time units is required by the user, then a different configuration is required--increase the capacity and/or use higher transmission speeds. In this example, because of the wide gap between the upper and lower bounds, the analyst would not be able to tell if a maximum of seventy time units for service time would be satisfied by the network. Only by applying the GCAA and computing the exact service time could this be known--in this case the requirement could not be met and the time and cost of running the GCAA would be an unprofitable expense. However, the closer the bounds are to each other, the higher the probability that running the GCAA will be a profitable expense.

If the given network is already in the form such that all overlapping two-arc three-node paths are non-queueing paths, then by calculating either the upper or lower bound the exact total service time is known--without using the GCAA. This is a time and money saver because an upper or lower bound can be computed in linear time while the GCAA requires an exponential time factor.

Leaving the generalized, systematic method of computing lower bounds as presented above, a tightening[1] of the bounds is possible in certain cases. For example, given a three-arc four-node network with parameters (10,5), (5,7) and (8,7) for arcs (1,2), (2,3) and (3,4), respectively, a tighter lower bound is possible by viewing path (1,2,3) as a one-arc two-node path with parameters (10,5) but with a starting time of five rather than zero. This, in turn, would now transform the path into case (G,L) (parameters (10,5) and (8,7)) so no queueing would occur in this path. The total service time computed via this method is thirty-three time units versus thirty for the original lower bound procedure.

Further analysis is needed to determine how much tighter the bounds are versus the original method. Study of numerous examples has not shown any great increase in accuracy, only a slight improvement. How conservative the bounds are depends on the interrelationship of the parameters of adjacent arcs in the network. It is impossible to predict a priori how conservative the bounds will be. However, the less the distance and capacity are changed by the bounding procedures, the better the bounds will be. In synthesizing a network from scratch, the analyst, by constructing his network to be non-queueing, can avail himself of the exact total service time by use of the bounding procedure alone at a very inexpensive cost.

---

[1]As suggested by Professor Pat Sterbenz.

Summary for calculating a lower bound on an N-1 arc N node network as in Figure 3.5:

Phase I - Transformation

(1) Set K = 1 (begin at the source node)

if N < 3 STOP (not a two-arc three-node path)

(2) Set $t_{i_{k+1},i_{k+2}}$ = minimum $(t_{i_k,i_{k+1}}, t_{i_{k+1},i_{k+2}})$

$C_{i_{k+1},i_{k+2}}$ = maximum $(C_{i_k,i_{k+1}}, C_{i_{k+1},i_{k+2}})$

(3) Set k = k+1 (move towards the destination node)

if k+2 > N go to Phase II (no more two-arc three-node networks)

(4) Go to step (2)

Phase II - Computation

(1) Lower bound =

$$\left(\left\lceil \frac{T}{C_{i_1,i_2}} \right\rceil - 1\right) \cdot t_{i_1,i_2} + \sum t_{i_m,i_n}$$

where i_m and i_n are adjacent nodes in the path

3.2.3 Extension to a General Network

Given a network with any number of nodes and arcs, as in Figure 3.7, this section will discuss the application of the bounding techniques for networks of N-1 arcs and N nodes to the general case. Each path from source to destination, assuming no node is visited twice in a path, can be viewed as an N-1 node N arc path. Thus, the bounding techniques of the previous two sections can be applied to each of these paths. After computing the upper and lower bounds for each path in the network and comparing the service time ranges, i.e., the upper and lower bounds, produced for those paths, one of the

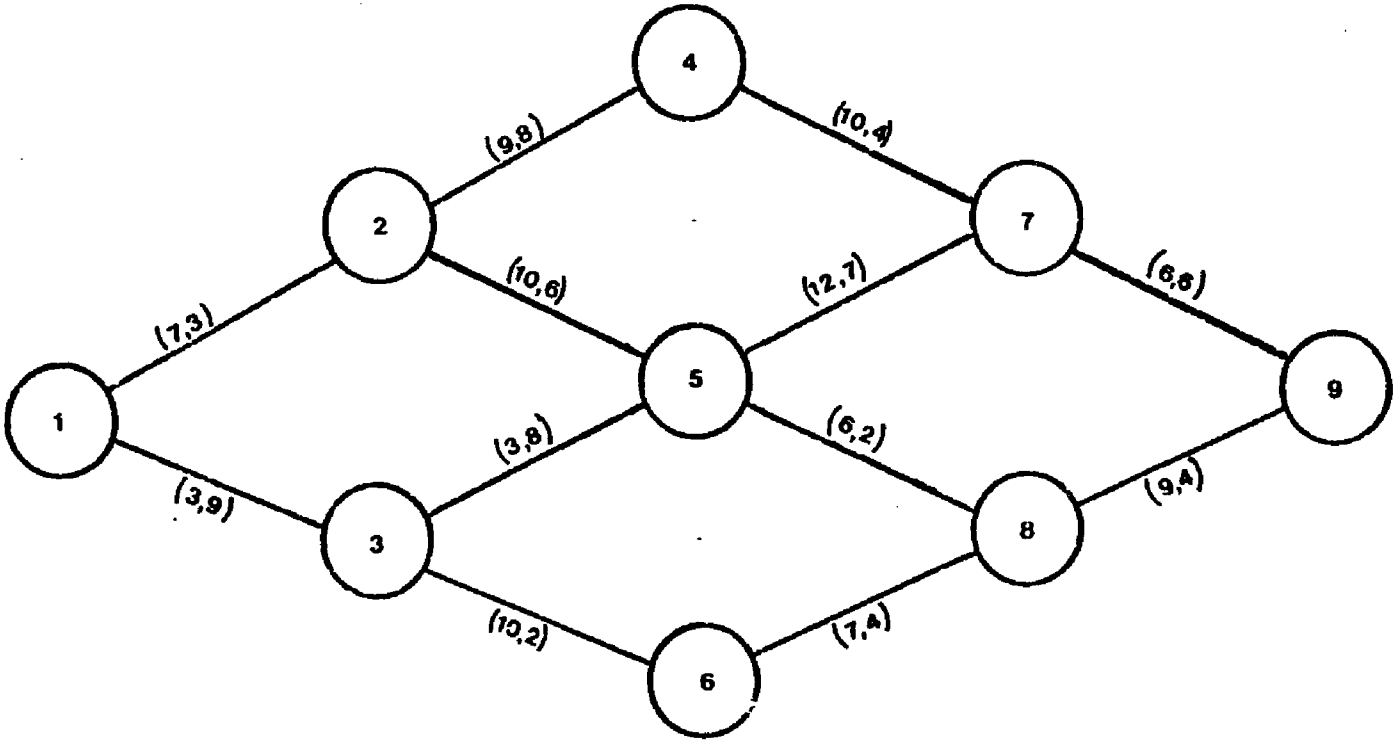

Figure 3.7. General directed network labeled with distance and capacity for each arc

following possiblities may occur. Assuming there are ranges that can satisfy the desired total service time parameter, if not a new configuration is required, one range may have an upper bound that is less than the requirement. For example, if the required service time is thirty-two time units and one path has a lower bound of ten and an upper bound of twenty-five, then it is without question the optimal path. Alternate routes can be chosen, if necessary, from those paths within whose ranges the desired total service time falls. (The following paragraphs will discuss selection of paths whose ranges contain the desired total service time.)

A second scenario is one in which no upper bound is less than the requirement but instead the requirement falls within certain ranges. Thus, if path A has a range of (30,35) and path B a range of (25,40), there is a better chance that path A will meet the requirement of thirty-four. Path A should therefore be tested first with the GCAA even though path B has a lower lower bound. The logic behind this is that since the upper bound of path A, thirty-five, is closer to the requirement of thirty-four than the upper bound of path B, forty, there is a higher probability that path A will satisfy the requirement, i.e., that the exact service time will be thirty-four or less. This is a conservative approach in that the path with the lower probability of failure is selected. Of course, if the budgetary and time constraints permit it, it may be worthwhile to test path B as well since it may provide a better solution if its exact value of service

time is less than the value for path A. The savings due to the increase in speed may offset the cost of running the GCAA. As this process is more of an art than a science, experience will increase the skill of the analyst in determining what the best option is. The more ranges there are to choose from, the more difficult the decision.

In transforming the network of Figure 3.7 from a queueing to a non-queueing configuration, the analyst can analyze each path from source to destination independently. For the eventual computerization of this procedure it may prove worthwhile to compute bounds in a dependent manner.

By viewing the paths independently, in the upper bound procedure, for example, those arcs that lead into nodes that have a fan-out of two or more are transformed twice. Node five of arc (2,5) in Figure 3.7 leads to arc (5,7) and arc (5,8). Thus, arc (2,5), when viewed as part of path (2,5,7) has a distance of twelve and a capacity of six. (Note that arc (5,7) was transformed in a previous step so that $d_{5,7}$ is now twelve and $C_{5,7}$ is six.) But when viewed as part of path (2,5,8), $d_{2,5}$ is ten and $C_{2,5}$ is two. This means that more than one value of distance and capacity has to be calculated and stored for each arc that is part of more than one path. This complicates the programming in that a varying number of distance and capacity values can be assigned to an arc rather than having a one-to-one relationship, i.e., one arc, one set of values. Arc (1,2) would require three sets of values since it is part of path (1,2,4,7,9), (1,2,5,7,9) and (1,2,5,8,9).

A modification will be introduced to maintain the above one-to-one relationship and thereby simplify the programming. In effect, it selects from all the arcs leading into the node the maximum distance and minimum capacity and transforms the arc leading into this node based upon this selection. In the above example, arc (2,5) would be transformed as follows. The distance is the maximum of ten, twelve and nine, the distances of arc (2,5), (5,7) and (5,8), respectively, which is twelve. The capacity of arc (2,5) is the minimum capacity of the same three arcs--two. Thus, twelve and two as the distance and capacity of arc (2,5) is a compromise between twelve and six (if arc (2,5) is looked at as part of path (2,5,7)) and ten and two (if arc (2,5) is in path (2,5,8)). What is really happening is that this new upper bound is the maximum possible of all three arcs. The transformation does not have to be done by comparing all three arcs in one step; instead if two are compared and the result of the comparison is compared to the third arc, the result will be the same. For example, if arcs (2,5) and (5,7) are compared, $d_{2,5}$ becomes twelve and $C_{2,5}$ becomes six. If these values are compared to arc (5,8), $d_{2,5}$ remains twelve but $C_{2,5}$ is decreased to two. Similar results would be obtained if the arcs were compared in a different order.

As in the upper bound scheme, the lower bound encounters a similar problem with intermediate nodes that have a fan-in of more than one. If the process of transforming the arc in question with regard to all of the fan-in

arcs is performed, then the one-to-one relationship will be maintained for lower bounds as well. The new values of distance and capacity would give the arc the lowest lower bound of all the choices.

This section has shown that the bounding techniques for N-1-arc N-node networks can be applied to networks of any number of nodes and arcs. The only restriction is that a path cannot contain the same node more than once. This will insure that the number of arcs in the path is one less than the number of nodes and thus fit into the N-1-arc N-node case. The bounding techniques provide the analyst with the tools to quickly obtain a handle on the network's performance under various traffic conditions. In some instances, the bounding techniques may provide a solution that will not require the use of the GCAA; in other circumstances, they can narrow down the number of paths that require further study via the GCAA to allow the analyst to optimally utilize the time and money involved in the design process.

CHAPTER 4

CONCLUSION

This thesis has examined the problem of computing the total service time in a store-and-forward packet-switched communication network. Two perspectives of the solution were presented:

1. An exact solution: an algorithm was presented and analyzed which computed the exact total service time for a message to traverse a path from source to destination.

2. Bounding techniques: procedures were introduced to compute upper and lower bounds in less time than the exact solution would require--to provide a quick handle on the total service time parameter.

Chapter 2 dealt with the computation of an exact value for service time. In section 2.2 the novel idea of the Cyclic Allocation Algorithm was introduced. This provided a simpler and faster method for computation of the exact service time over a typical discrete system simulation. Further, the speed-up technique allowed for a reduction in the number of computations needed to find the solution. The generalization of this method is found in section 2.3 where a recursive algorithm is formulated for the computation in N-1-arc N-node networks and the extension to general networks is presented. The simplicity and elegance of the algorithm make it a valuable tool for network designers.

Chapter 3 recognized that bounding techniques aid in

quickly detecting infeasible network configurations and thereby eliminating them from further consideration. This enabled the analyst to concentrate only on those designs that seemed promising. The crux of the proof for validity of the bounding procedures centered on an induction proof (section 3.1). This proof showed that four configurations of distance and capacity in a two-arc three-node network were queueless under any traffic conditions. The extension to N-1-arc N-node networks and the applicability to general networks was formulated in section 3.2. Thus, two vital tools were presented to enhance the process of network analysis and design--the General Cyclic Allocation Algorithm and the bounding techniques.

The bounding procedures are a novel contribution to the literature. Published literature is replete with exact solutions and heuristics for routing flows in networks to minimize average delay. However, these costly and time-consuming algorithms must be part of an iterative process to find a feasible solution that meets the user's requirement for total service time, thereby increasing the costs involved.

The upper and lower bound procedures arrived at as a result of this research can eliminate unnecessary testing of configurations. By allowing a quick and inexpensive method of weeding out infeasible designs, time and money can be saved and/or directed to the analysis of feasible configurations for the betterment of the final design. In addition, the bounding procedures produce the exact total service time

if the original network is non-queueing. This, at a great cost and time savings over exact solutions.

The exact solutions appearing in the literature compute all the flows in a network to minimize average delay time. The bounding procedures can be used in conjunction with any exact algorithm to first eliminate infeasible configurations. However, there is no simple way to indicate to the exact algorithms which paths, if any, to ignore. The GCAA can be used only on those paths the user wishes to test, thereby reducing even more the computation. Further, the GCAA with refinements can supersede direct simulation of the packet-by-packet flow on each arc for computation of total service time.

Thus, the novel bounding procedures not only allow weeding out of infeasible designs and a quick computation of exact total service time if the original network is non-queueing, but can be used in conjunction with published algorithms to speed up computation. The GCAA not only provides a straightforward and compact algorithm for computing exact total service time, in comparison to the algorithms in the literature, but with the stated refinements it can possibly surpass them.

The network designer can use the bounding procedures to his advantage in the synthesis of networks from scratch by using the non-queueing hypothesis as his guideline. By constructing configurations that are originally non-queueing, the bounding procedures will produce the exact total service

time. This will allow for optimal design with very inexpensive costs as compared to the exact algorithms in the literature. In general, the closer the transformed network is to the original, the better the bounds are.

Further research should be directed to determining the applicability and usefulness of alternate routes. This would entail the analysis of the merger of simultaneous packet flows at common nodes in the network. The analysis presented in this thesis treated the paths as independent units having in common only the source and destination nodes. If packet merging were to occur at intermediate nodes, additional queueing time might be incurred that would be unaccounted for in the GCAA or bounding schemes. The inclusion of this additional time in the current analysis requires further study. However, judging by ARPA network statistics, as the average utilization of transmission capacity is seven percent and ninety-seven percent of all messages are one packet long, packet merging does not pose an immediate problem to the network designer.

In addition, the extension of the GCAA and bounding procedures to multi-commodity flows and priority messages would greatly enhance their utility.

The analysis in this thesis was based on the assumption that the flow from one source was the only flow to be considered in determining capacity violations. With the introduction of multi-commodity flows, an arc's capacity can be affected by flows coming from more than one source. For example, in Figure 3.7, one path, (1,2,5,7,9), carries flow

from source node 1 to destination node 9, while path (2,5,8) carries flow from source node 2 to destination node 8. Arc (2,5) is common to both paths and its capacity of six units is affected by both flows. Given that the analysis only considers one flow at a time, there is no guarantee that if flows from both sources traverse arc (2,5) simultaneously that they might not exceed the capacity constraint of six. This must be accounted for in order to determine optimal routing for multi-commodity flows.

As in any typical network, certain messages are given higher priority than others to utilize arcs during transmission. These messages might be network-generated for updating local routing tables or user-generated for a real-time application that requires a very fast response time. In either case, lower priority packets traversing the network would be queued in order to let these higher priority packets transmit first. This extra delay is not accounted for in the current analysis and could affect the computation of total service time for both the higher and lower priority messages. Perhaps priority messages can be viewed as an extension of multi-commodity flows in that they, in effect, bring the capacity up to its maximum, thereby preventing lower priority messages from flowing on the same arc.

Continuing to be a major source of funds for packet-switching efforts, DARPA, the developer of the first packet-switched computer-communications network, has also provided impetus to developing packet-radio technology, which mates

broadcast radio with packet switching (Allan, 1981). This would be a computer-controlled network where internetwork, not just point-to-point, communication is needed regardless of whether or not line-of-sight conditions are available or any particular route between any two points is usable. In 1977, DARPA field-tested the packet radios in an experiment in the San Francisco Bay Area to assess the feasibility of a geographically distributed network of packet radios, controlled by one or more minicomputer-based stations. In 1978, the initial radios were replaced with packet radios having electronic counter-counter measure capabilities. The upgraded packet radios have verified the viability of the packet-radio concept in a tactical military environment.

Much other research and application is going on in packet-communication technology. Packet technology is seen as a basis for supporting integrated voice and data communication in a multiple network environment. DARPA officials feel that it is the only technology capable of achieving efficient use of communication capacity while supporting information exchange between people and computers. Commercial industry is also in on the ground floor of integrated network design, particularly Intecom and Northern Telecom, with plans for integrated business exchanges to carry voice and computer data at a substantial savings to the user over the currently separate exchanges needed to transmit these different types of information.

In conclusion, packet-switching technology will increasingly be turned to as the means to provide more economical and diversified communication services for both voice and data transmission in future network design.

APPENDIX A

LISTING OF THE COMPUTER PROGRAM FOR THE GCAA

```
GCAARUN: PROC OPTIONS(MAIN);

      DECLARE
       SERVICE FIXED DEC(9),
       (T,N) FLOAT,
       (C,D)(*) FLOAT CONTROLLED;

      DO WHILE('1'B);
       PUT SKIP LIST('ENTER NUMBER OF NODES');
       GET LIST(N);
       IF N = -1 THEN LEAVE;
       ALLOCATE C(N-1);
       ALLOCATE D(N-1);
       PUT SKIP LIST('ENTER CAPACITY');
       GET LIST(C);
       PUT SKIP LIST('ENTER DISTANCE');
       GET LIST(D);
       PUT SKIP LIST('ENTER NUMBER OF PACKETS');
       GET LIST(T);
       SERVICE = GCAA(T,N);
       PUT SKIP EDIT('SERVICE TIME : ',SERVICE)
       (COL(1),A,F(6));
     END;

GCAA:  PROC(T,N) RECURSIVE;
       DECLARE (T,N) FLOAT;

       IF N-1 = 1 THEN RETURN
       (D(N-1)*CEIL(T/C(N-1)));

       ELSE IF T - C(N-1) < 1 THEN
       RETURN(GCAA(T,N-1) + D(N-1));

       ELSE RETURN(MAX(GCAA(T,N-1),GCAA(T-C(N-1),N)) + D(N-1));
END GCAA;
END GCAARUN;
```

APPENDIX B

LISTING OF THE COMPUTER PROGRAM FOR THE EVALUATION OF UPPER AND LOWER BOUNDS

```
UPFLOW: PROCEDURE OPTIONS(MAIN);

        DECLARE
         (DIST,CAP,CONN,UC,UD,LC,LD)(*,*) FIXED BIN CONTROLLED,
          NEXT(*) FIXED DEC(3) CONTROLLED,
         PATH(20,*) FIXED DEC(3) CONTROLLED,
         LIST(*) FIXED DEC(3) CONTROLLED,
         (PNUM,LASTX,NDX,HIT) FIXED BINARY,
         FILEIN FILE INPUT;

         ON ENDFILE(FILEIN) GO TO NEXTSTEP;

         GET FILE(FILEIN) LIST(LIM);
         ALLOCATE DIST(LIM,LIM);
         ALLOCATE CAP(LIM,LIM);
         ALLOCATE CONN(LIM,LIM);
         ALLOCATE UC(LIM,LIM);
         ALLOCATE UD(LIM,LIM);
         ALLOCATE LC(LIM,LIM);
         ALLOCATE LD(LIM,LIM);
         ALLOCATE NEXT(LIM-1);
         ALLOCATE PATH(20,LIM);
         ALLOCATE LIST(LIM);
         LIST = 0;  LIST(1) = 1;
         DIST=0; CAP=0; CONN=0;
         UC=0; UD=0; LC=0; LD=0;
         PATH = 0;  PATH(*,1) = 1;
         PNUM = 1;  NEXT = 0;

         DO WHILE('1'B);
          GET FILE(FILEIN) LIST(I,J,DIST(I,J),CAP(I,J));
          CONN(I,J) = 1;
         END;
```

```
NEXTSTEP:
        UD,LD= DIST;   UC,LC= CAP;

/*      COMPUTE LOWER BOUNDS */
        DO I = 1 TO LIM;
         DO J = 1 TO LIM;
          IF CONN(I,J) = 1 THEN
          DO;
           DO K = 1 TO LIM;
            IF CONN(J,K) = 1 THEN CALL LOW_BOUND(I,J,K);
           END;
          END;
         END;
        END;

/*      COMPUTE UPPER BOUNDS */
        DO K = LIM TO 1 BY -1;
         DO J = LIM TO 1 BY -1;
          IF CONN(J,K) = 1 THEN
          DO;
           DO I = LIM TO 1 BY -1;
            IF CONN(I,J) = 1 THEN CALL UP_BOUND(I,J,K);
           END;
          END;
         END;
        END;

/*     PRINT THE ORIGINAL VALUES OF DISTANCE AND CAPACITY
       FOR EACH ARC AS WELL AS THE UPPER AND LOWER BOUNDS  */

       PUT SKIP(2) EDIT('I','J','OD','OC','UD','UC','LD','LC')
      (COL(4),A,COL(8),A,(6)(X(2),A(2)));
      DO I = 1 TO LIM;
       DO J = 1 TO LIM;
        IF CONN(I,J) = 1 THEN PUT SKIP EDIT
        (I,J,DIST(I,J),CAP(I,J),UD(I,J),UC(I,J),LD(I,J),LC(I,J))
        (COL(1),(8)F(4));
       END;
      END;
```

```
/*    USING THE CONNECTIVITY MATRIX 'CONN' DETERMINE ALL THE
      PATHS IN THE NETWORK AND STORE THEM IN ARRAY 'PATH' FOR
      USE IN THE COMPUTE STEP BELOW */

      I = 1;    LNDX = 1;      II = 1;
MAINLOOP:
      DO WHILE('1'B);
       HIT = 0;
LOOPJ:
      DO J  = II+1 TO LIM;
       IF CONN(I,J) = 1 THEN
       DO;
        HIT = 1;
        LNDX = LNDX + 1;
        LIST(LNDX) = J;
        LEAVE LOOPJ;
       END;
      END;

      IF HIT = 1 THEN
      DO;
       NEXT(I) = J;
       IF J < LIM THEN
       DO;
        II = I;
        I = J;
       END;
       ELSE IF J = LIM THEN
       DO;
        NDX = 1;  K = 1;
        DO WHILE(K < LIM);
         NDX = NDX + 1;
         LASTK = K;
         K = NEXT(K);
         IF K > 0 THEN PATH(PNUM,NDX) = K;
        END;
        NEXT(LASTK) = 0;
        PUT SKIP EDIT('PATH ',PNUM,' : ',
        (PATH(PNUM,JJ) DO JJ=1 TO LNDX))
        (COL(1),A,F(2),A,(LNDX)F(3));
```

```
    PNUM = PNUM + 1;
    LNDX = LNDX - 2;
    II = LIST(LNDX+1);
    I = LIST(LNDX);
   END; /* IF J = LIM */
  END; /* IF HIT = 1 */
  ELSE DO;
   IF I = 1 & J >= LIM THEN LEAVE MAINLOOP;
   ELSE IF J >= LIM THEN
   DO;
    NEXT(I) = 0;
    II = I;
    LNDX = LNDX - 1;
    I = LIST(LNDX);
   END;
   ELSE DO;
    II = J;
    I = J;
   END;
  END;
END; /* MAINLOOP */
PUT SKIP LIST('ALL PATHS COMPUTED');
```

```
/*     THE USER ENTERS A VALUE FOR THE NUMBER OF PACKETS
       TO BE SHIPPED FROM SOURCE TO DESTINATION (NODE 1 AND
       NODE 'LIM' RESPECTIVELY) AND THE PROGRAM COMPUTES THE
       UPPER AND LOWER BOUNDS FOR EACH PATH */

COMPUTE:
      DO WHILE('1'B);
       PUT SKIP(2) LIST('ENTER THE NUMBER OF PACKETS');
       GET LIST(PACKETS);
       IF PACKETS  = 0 THEN LEAVE COMPUTE;
       DO I = 1 TO PNUM-1;
        MINCUP = 9999;  MINCLO = 9999;
        TSTUP = 0;      TSTLO = 0;
        DO J = 1 TO 9 WHILE(PATH(I,J+1) > 0 );
         K = PATH(I,J);    L = PATH(I,J+1);
         TSTUP = TSTUP + UD(K,L);
         TSTLO = TSTLO + LD(K,L);
         IF UC(K,L) < MINCUP THEN MINCUP = UC(K,L);
         IF LC(K,L) < MINCLO THEN MINCLO = LC(K,L);
        END;
        K = PATH(I,1);    L = PATH(I,2);
        TSTUP = TSTUP + (CEIL(PACKETS/MINCUP)-1)*UD(K,L);
        TSTLO = TSTLO + (CEIL(PACKETS/MINCLO)-1)*LD(K,L);
        PUT SKIP EDIT('PATH ',I,'  UPPER BOUND : ',TSTUP,
        ' LOWER BOUND : ',TSTLO)(COL(1),A,F(3),(2)(A,F(6)));
       END;
      END;

UP_BOUND: PROCEDURE(START,MID,FINISH);
           DECLARE (START,MID,FINISH) FIXED BIN;
           UD(START,MID) = MAX(UD(START,MID),UD(MID,FINISH));
           UC(START,MID) = MIN(UC(START,MID),UC(MID,FINISH));
END UP_BOUND;

LOW_BOUND: PROCEDURE(START,MID,FINISH);
          DECLARE (START,MID,FINISH) FIXED BINARY;
          LD(MID,FINISH) = MIN(LD(START,MID),LD(MID,FINISH));
          LC(MID,FINISH) = MAX(LC(START,MID),LC(MID,FINISH));
END LOW_BOUND;
END UPLOW;
```

REFERENCES

Allan, R. 1981. "Military Electronics: Systems," Electronic Design, pp. 87-94.

Best, M. 1975. "Optimization of Nonlinear Performance Criteria Subject to Flow Constraints," Proc. 18th Midwest Symposium on Circuits and Systems, Concordia University, Quebec, Canada, pp. 438-443.

Brown, C.W. 1975. "Adaptive Routing and Resource Allocation," Ph.D. dissertation (Electrical Engineering), Polytechnic Institute of New York.

Brown, C.W., and Schwartz, M. 1975. "Adaptive Routing in Centralized Computer-Communication Networks," Proc. IEEE International Conference on Communications, San Francisco, pp. 47-12 to 47-16.

Burke, P.J. 1956. "The Output of a Queueing System," Operations Research, Vol. 4, pp. 699-704.

Cantor, D.G., and Gerla, M. 1974. "Optimal Routing in a Packet-Switched Computer Network," IEEE Trans. on Computers, C-23, No. 10, pp. 1062-1069.

Chretien, G.J., König, W.M., and Rech, J.H. 1963. "The Sita Network, Summary Description," Computer-Communication Network Conference, University of Sussex, Brighton, U.K.

Danzig, G.B. 1963. Linear Programming and Extensions. Princeton, N.J.: Princeton University Press.

Everett, R.R., Zraket, C.A., and Benington, H.D. 1957. "SAGE: A Data Processing System for Air Defense," Proceedings of Eastern Joint Computer Conference.

Fano, R.M. 1965. "MAC System: The Computer Utility Approach," IEEE Spectrum.

Frank, H., and Chou, W. 1971. "Routing in Computer Networks," Networks, Vol. 1, No. 2, pp. 99-112.

Frank, H. 1979. "A Decade of Birth," Computerworld (special edition, Surveying the '70s As We Enter the '80s), Vol. XIII, No. 53/Vol. XIV, No. 1.

Fratta, L., Gerla, M., and Kleinrock, L. 1973. "The Flow Deviation Method: An Approach to Store-and-Forward Computer Communication Network Design," Networks, Vol. 3, pp. 97-133.

Fultz, G.F., and Kleinrock, L. 1971. "Adaptive Routing Techniques for Store-and-Forward Computer-Communication Networks," Proceedings IEEE International Conference on Communications, Montreal, pp. 39-1 to 39-8.

Hadley, G. 1964. Non Linear and Dynamic Programming. Reading, Mass.: Addison-Wesley.

Kleinrock, L. 1964. "Communication Nets, Stochastic Message Flow and Delay," New York: McGraw-Hill. Out of print. Reprinted by Dover Publications, 1972.

_____. 1975. Queueing Systems, Vol. I: Theory, New York: Wiley Interscience.

_____. 1976. Queueing Systems, Vol. II: Computer Applications, New York: Wiley Interscience.

Kobayashi, H. 1974. "Applications of the Diffusion Approximation to Queueing Networks, part II," Journal of the ACM, pp. 459-69.

Morrisey, J.H. 1965. "The QUICKTRAN System," Datamation.

Perry, M.N., and Plugge, W.R. 1961. "American Airlines SABRE Electronic Reservation Systems," AFIPS Conference Proceedings, Western Joint Computer Conference, No. 19.

Popell, S.D., et al. 1966. Computer Time Sharing. Englewood Cliffs, N.J.: Prentice-Hall.

Prosser, R.T. 1962. "Routing Procedures in Communication Networks, part I: Random Procedures," IRE Trans. on Communication Systems, CS-10, No. 4, pp. 322-329.

Roberts, L.G. 1967. "Multiple Computer Networks and Inter-Computer Communications," Proc. of the ACM Symposium on Operating Systems, Gatlinburg, Tennessee.

Roberts, L.G., and Wessler, B.D. 1970. "Computer Network Development to Achieve Resource Sharing," Spring Joint Computer Conference, AFIPS Conference Proceedings, Vol. 36, pp. 543-599.

Rudowsky, I., and Rootenberg, J. 1978. "An Upper Bound for the Total Service Time in Packet-Switched Store-and-Forward Communication Networks," Proceedings Computer Networking Symposium, Gaithersburg, Md., pp. 26-32.

_____, and _____. 1979. "Queueing Effects in the Solution of Constrained Store-and-Forward Network Flow Problems," Int. J. Systems Sci., Vol. 10, No. 7, pp. 775-782.

Schwartz, M., and Cheung, C.K. 1976. "The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks," IEEE Trans. on Communications, COM-24, No. 4, pp. 449-456.

Schwartz, M. 1977. Computer Communication Network Design and Analysis. Englewood Cliffs, N.J.: Prentice-Hall.

Silk, D.J. 1969. "Routing Doctrines and Their Implementation in Message Switching Networks," Proceedings IEE, London, 116, No. 10.

Tomlin, J.A. 1966. "Minimum Cost Multicommodity Network Flows," Operations Research, Vol. 14, pp. 45-47.

Walden, D.C. 1975. "Experiences in Building, Operating and Using the ARPA Network," Second USA-Japan Computer Conference, Tokyo.