Technical Report 839

NETWORK DESIGN TOOL FOR EHF SATELLITE COMMUNICATIONS NETWORKS

Sally Norvell
G. J. Brown

June 1983

Final Report

Approved for public release; distribution unlimited

NOSC

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152

NOSC

NAVAL OCEAN SYSTEMS CENTER, SAN DIEGO, CA 92152

AN ACTIVITY OF THE NAVAL MATERIAL COMMAND

JM PATTON, CAPT, USN
Commander

HL BLOOD
Technical Director

ADMINISTRATIVE INFORMATION

The work covered by this report was done during the period of January to September 1982 under the Independent Research/Independent Exploratory Development program.

Released by
M.S. Kvigne, Head
Communications Research
and Technology Division

Under authority of
H.D. Smith, Head
Communications Systems
and Technology Department

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
NOSC Technical Report 839 (TR 839)

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
NETWORK DESIGN TOOL FOR EHF SATELLITE COMMUNICATIONS NETWORKS

5. TYPE OF REPORT & PERIOD COVERED
Final Report
January to September 1982

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Sally Norvell, G.J. Brown

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Ocean Systems Center
San Diego, CA 92152

10. PROGRAM ELEMENT, PROJECT TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

12. REPORT DATE
June 1983

13. NUMBER OF PAGES
158

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Network Design Tool (NDT)
Network Switching Center (NSC)
GRINDER (Graphical Interactive Network Designer)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the design concept of the Network Design Tool. The design is based on currently available modeling and analysis techniques and the techniques, along with available network design tools, are surveyed. The limitations of the available design tools are discussed as well as the two major network design problems, the topological design/optimization problem and the protocol validation problem. The Network Design Tool for the EHF satellite communications networks is presented.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ACRONYMS AND GLOSSARY

ACEX	Area-Code/Exchange
ACT	Analyst Control Terminal
ACUT	Automated Cut Saturation Algorithm
AHPL	A discrete system, descriptive/design language
ALGOL	Algebraically Oriented Language (algorithmic language)
APL	Advanced Programming Language
ARPANET	Advanced Research Projects Agency Network
ASG	Assign
AUTODIN	Automated Digital Information Network
BGS	Developers of the POD System
BKBN	Backbone Node
BXC	Branch Exchange
CA	Capacity Assignment
CBE	Concave Branch Elimination
CDMA	Code Division Multiple Access
CFA	Capacity and Flow Assignment
CNTR	Center
COM	Center of Mass
CONC	Concentrator
CPU	Central Processing Unit
DAMA	Demand Assigned Multiple Access
DASG	Deassign
DCDL	A discrete system, descriptive/design language
DFSS	Data Flow Simulation System
DYNAMO	A simulation programming language

ACRONYMS AND GLOSSARY (Continued)

EF	Extremal Flow Algorithm
EHF	Extremely High Frequency
FA	Flow Assignment
FD	Flow Deviation
GASP	Fortran-based simulation package
GLAD	Component of GRINDER
GMB	Graph Model of Behavior
GMC	Graph Model of Computation
GPSS	General Purpose System Simulation
GRINDER	Graphical Interactive Network Designer
GRST	Ground Station node
HIER	Component of GRINDER
HOME	Homing Point
IMP	Interface Message Processor
ISI	Information Science Institute (USC)
ISIA	Information Science Institute Computer
ISO-OSI	International Standards Organization-Open Systems Interconnection
ISP	A discrete system, descriptive/design language
LD	Lagrangian Decomposition
LL	Line Layout
LOC	Location Identifier
MIDAS	A simulation programming language
MIND	Modular Interactive Network Design System
MIT	Massachusetts Institute of Technology
MLSS	Multipoint Line Simulation System

ACRONYMS AND GLOSSARY (Continued)

MPDL	A discrete system, descriptive/design language
MPL	Multischedule Private Line
MST	Minimum Spanning Tree
NAC	Network Analysis Corporation
NDT	Network Design Tool
NSC	Network Switching Center
PHDB	Project History Data Base
PLT	Piecewise Linear Tariff Structure
POD	Performance Oriented Design
SARA	System Architects' Apprentice
SATL	Satellite node
SC	Subscriber computer
SIMULA	A simulation programming language
SIMSCRIPT	A simulation programming language
SOL	Sequence Orders List
TCFA	Topology, Capacity and Flow Assignment
TCU	Terminal Control Unit
TDMA	Time Division Multiple Access
TLPK	Telpak (speed dependent tariff structure)
TOPO	Topological Design and Analysis
UCLA	University of California - Los Angeles
VANS	Value Added Network Simulator

CONTENTS

CONTENTS (Continued)

ILLUSTRATIONS

ILLUSTRATIONS (Continued)

TABLES

CHAPTER 1

INTRODUCTION

PURPOSE AND GOALS

This document describes the design concept of the "Network Design Tool." The Network Design Tool (NDT) is a collection of analytical techniques, algorithms and simulation methods that may be used to characterize the performance of a computer communication network. Much work has been done over the past several years in network performance analysis and many techniques have been developed or proposed. Each of these methods applies to a particular aspect of the network design and is based on a particular modeling point of view. We define the computer communication network and then describe the different ways the network may be modeled. Each network model is related to the particular design problem being addressed. The various analytical approaches are briefly described and their relationship to the network models discussed.

Chapter 2 is a survey of the major approaches to specific network design problems while Chapters 3 and 4 discuss two fairly well defined areas of network analysis: topological design/optimization and protocol validation. Chapter 5 is a survey of network design tools presently available locally or on the Advanced Research Projects Agency Network (ARPANET). Finally, Chapter 6 presents an outline of the NDT specification.

NETWORK DESCRIPTION

A discussion of network analysis is impossible without first defining what a network is. A network may mean entirely different things to different people. One person may view a network as a switching scheme where another may view it as an access scheme. For our purposes we define a computer communications network as a set of two or more computer nodes interconnected by communication links to provide services for a host function. A node may be a host that is responsible for command and control functions or it may be responsible only for communication functions. We are concerned here with only the communication functions.

A network may be viewed many ways, depending on the viewer's particular interest. A manager may be most interested in how and where the network nodes are located, whereas a designer may be concerned about the switching technique. Networks may be classified by services provided, routing schemes, access methods, switching techniques, topology capacity or communication characteristics. The computer communication network classifications are summarized in Table 1.[1] The table shows that the network design problem actually

Viewpoint	Viewer's Interest	Model Categories
Functional	Type of host access service provided — terminal access or batch, user to one host, user to any host, host to host.	Remote access net Value added net Mission oriented net
Switching (Designer)	Node interconnection technique	Circuit switched Message switched Packet switched Hybrid switched
Manager (Facilities Placement)	Topology, capacity, congestion and reliability	Centralized Decentralized Distributed
Operational 1	Routing and flow control	Deterministic Stochastic Adaptive/non-adaptive
Communication	Communicating components and communication characteristics	Resource sharing Distributed Computations Remote Communications
Operational 2	Access Control	Centralized Decentralized Adaptive
Hybrid 1	Routing and Topology	
Hybrid 2	Routing and Access	
Hybrid 3	Access and Switching	

Table 1. Computer network classifications.

consists of several separate problems that are coupled together, namely:

1. Topological design (including channel capacity and flow)
2. Switching design
3. Access control
4. Routing
5. Flow control

Figure 1 illustrates how these elements may be treated in a top down fashion, from the highest level viewpoint (topology) to the lowest (protocol validation). Of course these elements are not so neatly separated and, in fact, they must be integrated at some point in the design. The figure also shows that the designs must be put into the form of well defined procedures (protocols) that are then checked for correct operation using formal validation techniques.

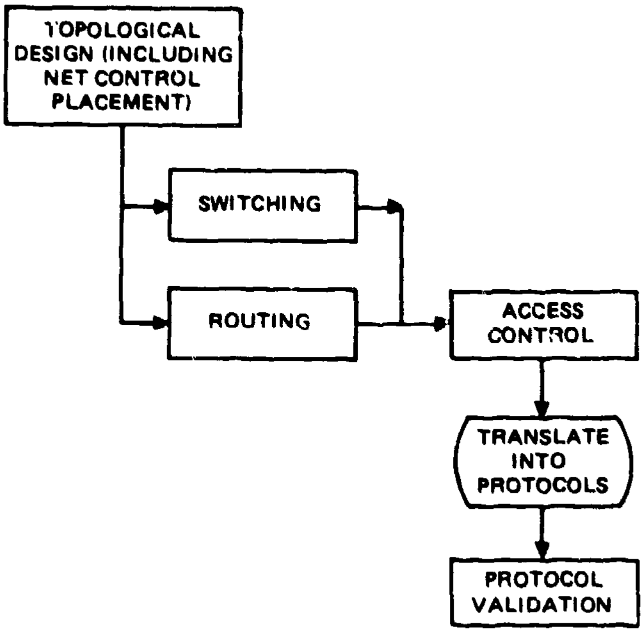


Figure 1. Components of the network design problem.

NODE DESCRIPTION

The topological design and switching design problems can assume a fairly high level model where each node is just a source, sink or redirector of traffic, as determined by a node genie. The lower level design problems and generation of protocols require a more detailed description of the node itself. This is provided by the layered architecture suggested by the International Standards Organization-Open Systems Interconnection (ISO-OSI) reference model.[2] The model, shown in Figure 2, defines an architecture that separates the protocols into a logical order (layers) where a layer communicates only with the layer immediately above and below it. The communication aspects of the node are contained in layers 1 through 4 (communications subnetwork). Switching, routing and access control are contained in these layers.

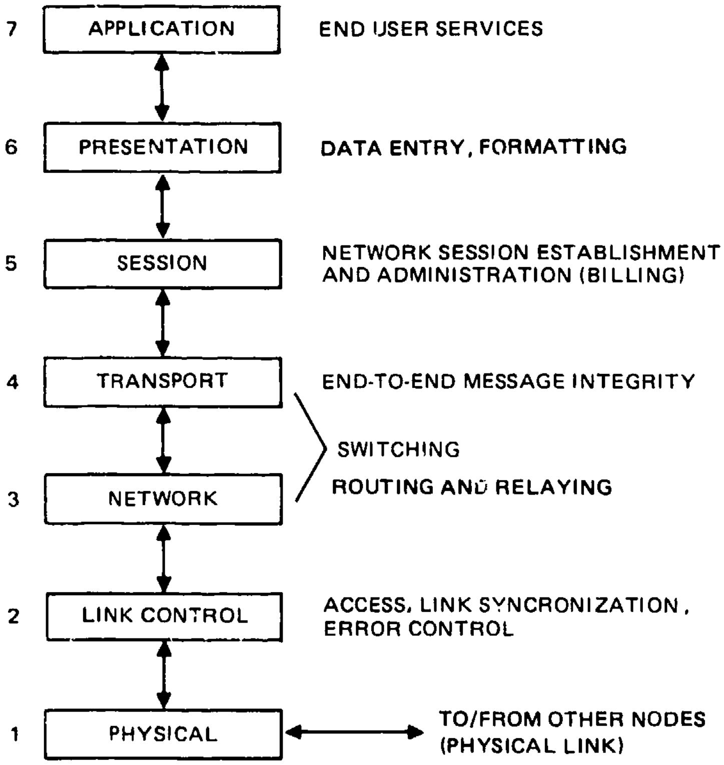

7
APPLICATION
END USER SERVICES
6
PRESENTATION
DATA ENTRY, FORMATTING
5
SESSION
NETWORK SESSION ESTABLISHMENT AND ADMINISTRATION (BILLING)
4
TRANSPORT
END-TO-END MESSAGE INTEGRITY
SWITCHING
ROUTING AND RELAYING
3
NETWORK
2
LINK CONTROL
ACCESS, LINK SYNCRONIZATION, ERROR CONTROL
1
PHYSICAL
TO/FROM OTHER NODES (PHYSICAL LINK)

Figure 2. ISO-OSI reference model.

The value of the layered architecture may be seen by the following example using a packet routing scheme. To support a packet switching scheme, the node must perform a variety of tasks including message reconstruction, packet acknowledgement and channel access. Using the layered architecture, the routing algorithm by itself may be considered, as in Figure 3. The network layer sees only packets coming from or going to the transport layer. It communicates with the routing protocols (peer level communications) at other nodes on a virtual channel that actually corresponds to the lower layers and the physical channel.

The network design techniques presented in this report are best applied to systems that follow the layered architecture.

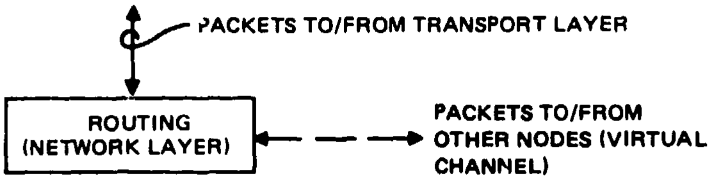

PACKETS TO/FROM TRANSPORT LAYER
ROUTING (NETWORK LAYER)
PACKETS TO/FROM OTHER NODES (VIRTUAL CHANNEL)

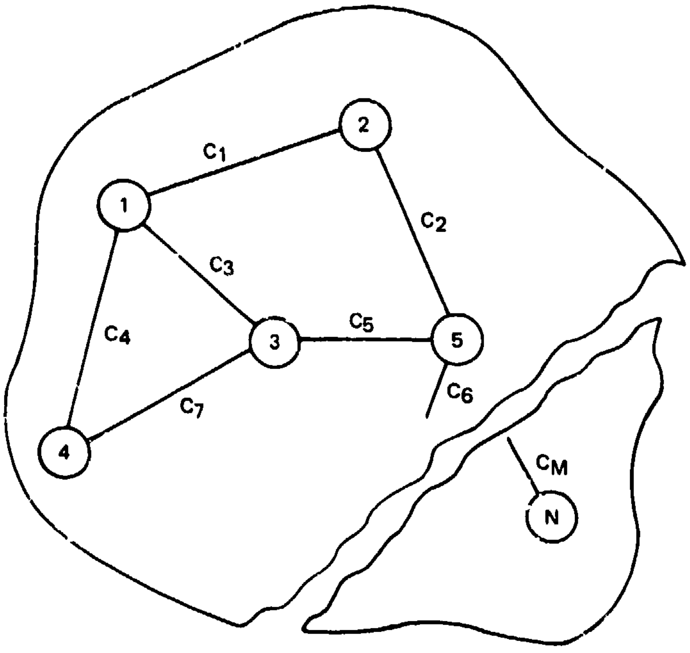
Figure 3. Isolation of routing algorithm using the layered architecture.

DEFINITION OF ANALYSIS AND DESIGN PROBLEMS

The previously mentioned design problems are described next in more detail.

Topological Design

Figure 4 shows the usual network model for the topological design problem. The model consists of N nodes, which may be used to switch packets or messages or to form circuits, and M circuits, which are noiseless and bidirectional with capacity C_i bits per second. The nodal processing time is constant and is usually assumed negligible. Messages may originate external to the network or at another node. All messages are assumed to have lengths independently drawn from an exponential distribution. The time to service a message depends on its length and the fixed parameters of the channels.



NODAL PROCESSING TIME = K

Figure 4. Network model.

There are three basic design parameters for this network:

1. Selection of the channel capacities (C_i)
2. Selection of the channel flows (λ_i)
3. Placement of the nodes (topology)

The performance criterion is the average message delay, T. There is also a cost constraint, D.

The optimization problems are summarized in Table 2. These four problems have been solved to various degrees, depending on the form of the cost function. Software packages are available (see Chapter 5) for performing the optimizations.

Switching Design

Circuits from a source node to a destination node may be formed by switching intermediate segments to form one continuous route. This switching

Problem	Given	Minimize	With Respect To	Under Constraint
1. Capacity Assignment	Flow $\{\lambda_i\}$ and Network topology	Delay (T)	Channel Capacities $\{C_i\}$	Cost $D = \sum_{i=1}^{M} d_i(C_i)$ $d_i(C_i)$=Cost Channel i
2. Flow Assignment	Capacities $\{C_i\}$ and network topology	T	Flows $\{\lambda_i\}$	
3. Capacity and Flow Assignment	Network topology	T	$\{C_i\}$ and $\{\lambda_i\}$	D
4. Topology, Capacity and Flow Assignment		T	Topological design, $\{C_i\}$, $\{\lambda_i\}$	D

Table 2. Network optimization problems.

may be done on a long term basis (the duration of the service) as in circuit switching. The route may be reconfigured on every message (message switching) or more frequently (packet switching). More importantly, packet switching allows a mix of message traffic to share a channel, whereas in circuit switching a short message may have to wait while a long message is being transmitted. A common tradeoff is between circuit switching and packet switching.[3] With the proper assumptions (see Reference 3), an approximate solution to this problem may be found by iteratively solving a nested sequence of related-integer linear capacity flow equations. The switching design depends upon many factors including traffic statistics, channel quality, delay requirements, switching node complexity, buffer sizes and channel cost. The parameters involved are packet lengths, buffer lengths and message delay. Figure 5 outlines the switching alternatives.

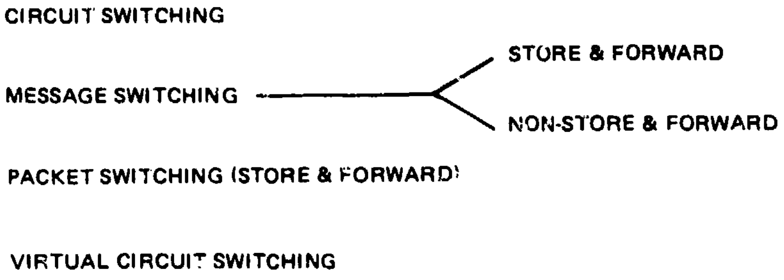


Figure 5. Switching schemes.

Access Design

A discussion of switching naturally leads to the access design problem, i.e., the question of how multiple users share a single channel resource. Literally hundreds of access techniques and variations have been described in the literature. The usual design tradeoff shows traffic intensity versus packet delay (for packet switching), or traffic intensity versus blocking probability (for message or circuit switching) and throughput versus offered traffic load. The performance analysis has been done using both simulation and analytical techniques. The theory of probability and stochastic processes and queueing theory have been used to solve problems when certain assumptions have been made. The assumptions usually are: all users are statistically identical, independent Poisson sources and buffers are of infinite capacity. Reference 4 provides a good survey of access schemes and analysis. The reference organizes the schemes as outlined in Figure 6.

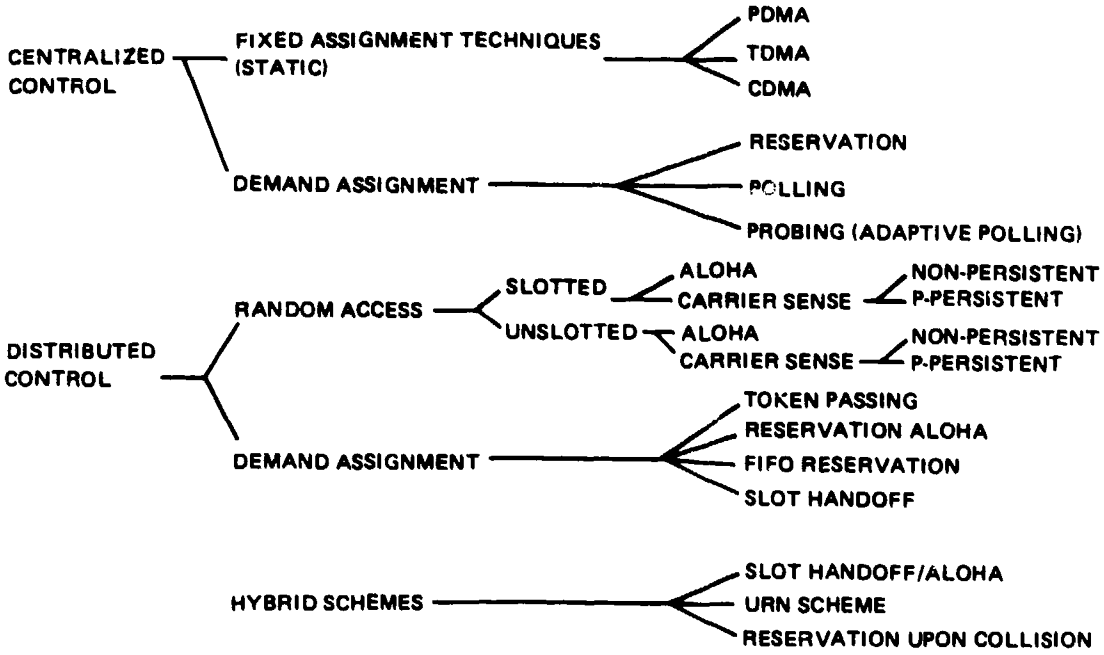


Figure 6. Access scheme tree.

Routing Design

Although the routing design affects the access, switching and topology designs, it is treated somewhat independently for simplicity. The routing problem is to select the best route (least delay) from source to destination. If a flow and channel capacity optimization has been performed, then the routing algorithm must be designed to fit that optimization. The topology and the distribution of control may constrain the routing design. A diagram of this relationship is shown in Figure 7 (from Reference 1). Since most routing algorithms are derived in an ad hoc fashion and involve transactions between multiple nodes, analytical methods are difficult to use and simulation is the usual method for analysis.

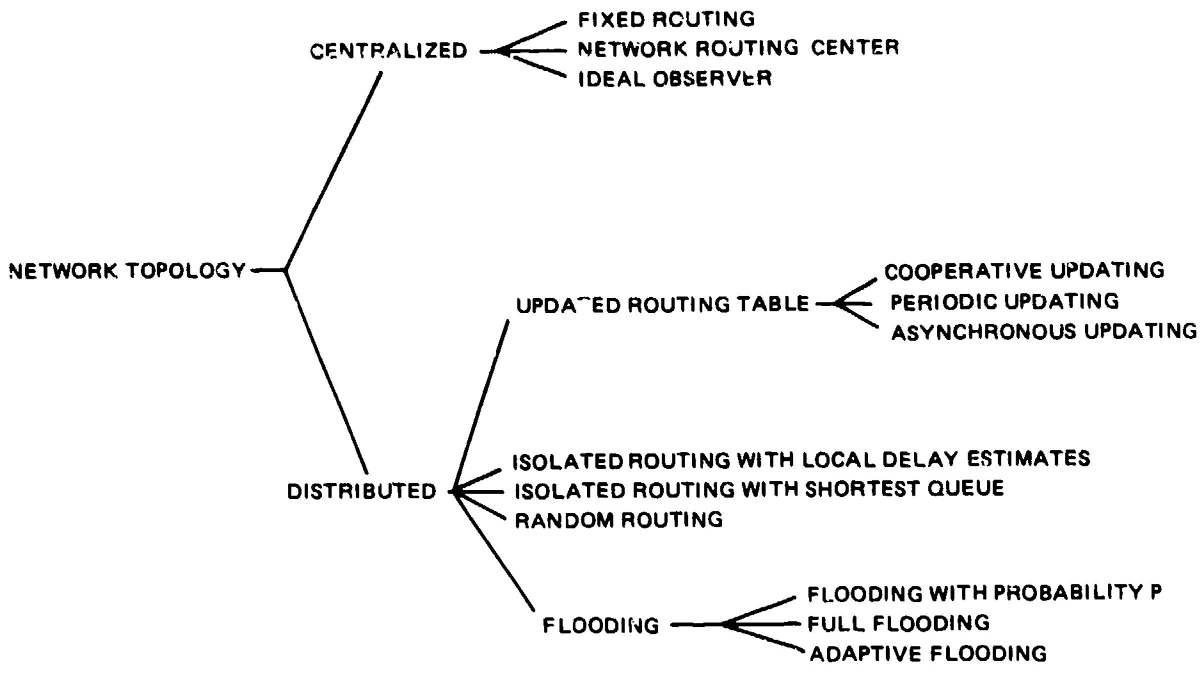


Figure 7. Routing scheme tree.

An analytical technique that has had some success is the use of graph theory[5] for both the design and analysis of optimum fixed routing schemes.

Flow Control

A packet switched network must control the rate of admission of packets into the network to avoid overly congesting the network. These rules are known as flow control procedures. Flow control procedures are local, limiting packet admission at a node and global, limiting the total number of packets in the network. Flow control, like routing, is difficult to analyze and simulation is often used. Figure 8 shows the basic flow control alternatives.

THE REAL NETWORK

There are several design issues that are not addressed by analysis techniques. Generally they require custom designed simulations to address their unique requirements. These issues arise in real communication networks,

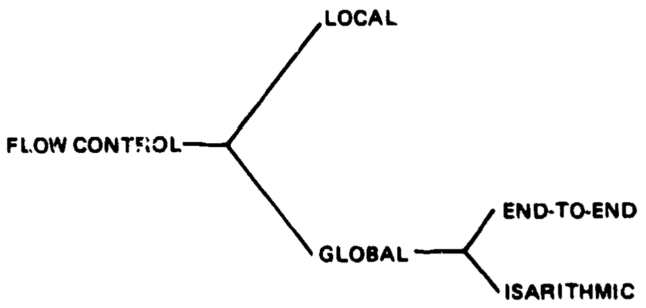


Figure 8. Flow control schemes.

especially military communication networks. Those design issues typically ignored include the following:

1. Dynamic topologies
2. Node destruction
3. Jamming
4. Noisy links
5. Transient results
6. Multilevel priorities
7. Preemption
8. Control transfer (survivability)

The available network design packages, described in Chapter 5, do not have the capabilities to handle these design issues. The proposed Network Design Tool, described in Chapter 6, is designed to address these complex issues.

CHAPTER 2

SPECIFIC APPROACHES TO NETWORK DESIGN AND ANALYSIS

Several modeling techniques are used to study computer communication networks. First, there are theoretical models that are often used as a basis for analyzing computer networks. Then, since most work in computer networks is based on a theoretical queueing theory model, there are queueing theory and models based on networks of queues. Another set of powerful analytical tools is provided by the theory of stochastic processes. This includes renewal theory, Markov chain theory, semi-Markov and regenerative processes and Markov decision theory. Next, there is the potential application of graph theory to network design and analysis. We also discuss here finite state descriptions.

Since theoretical models may not always have solutions which can be obtained in finite time, approximations to theoretical models and solution techniques to these approximations have been developed. Several approximation and solution techniques are presented here. The next set of models is not analytical. This set includes simulation techniques, hybrid techniques and modular techniques. Also included are on-line simulation, a testbed approach and an approach which uses parallel processors to emulate communication networks. These approaches are tabulated in Table 3.

QUEUEING MODELS

The growing complexity of computer systems has motivated the development of analytic tools to investigate computer system performance. A queueing network model is the most widely used approach in the design and analysis of communication networks. If desired, very complex systems can be characterized by a few parameters if certain simplifying assumptions are made. Anticipated or proposed changes to the system can be represented by adjusting some of the parameter values. Queueing network models have proven effective in predicting the effects of changes on system performance.

The accuracy of analytic models and the amount of detail of a system they can capture are restricted by their computational solution algorithms.

Approach	Net Model	Method	Purpose	References
Queueing Theory	Net of queues	Analytical	Flow & topology design	6,7,8,9
Stochastic Modeling	Markov process	Markov chain theory	Protocol analysis	
	Semi-Markov and regenerative process		Protocol analysis	
Graph Theory	Graph: N Nodes connected by L arcs	Graph theory (PRIM algorithm, Esau-Williams, Branch and Bound, etc.)	Line cost minimization, shortest routes, connection reliability	10
Net Models	Graph models consisting of transitions and places interconnected by edges. Uses tokens flowing from place to place across transitions	Petri net simulator implemented in APL	Detect deadlocks, traps, and control-flow anomalies in protocols	
Finite State	Finite state model		Protocol analysis	11
Approximation Techniques	Iterative techniques Diffusion approximation Decomposition techniques	Enable user to obtain approximate solution to complex analytical problem		
Hybrid Schemes	Analytic/simulation	Combine some analytic techniques with simulation		
Modular (Vans)	Divide net activities into isolated protocols	Software simulation using modular S/W design. Module = Protocol	Full net modeling and protocol evaluation	12,13,14,15

Table 3. Network design/analysis approaches.

Approach	Net Model	Method	Purpose	References

Discrete Event Simulation		Special purpose simulation language - GPSS, SIMSCRIPT, SIMULA	Full network or protocol evaluation
			SIMSCRIPT = Interactive
			SIMULA = Batch
				16,17

		General purpose languages		
		Descriptive/design languages (ISPS, AHPL, DCDL, MPDL)	Accurate structure and behavior modeling. Hardware/software design	
Continuous Event Simulation	Variables change smoothly - mathematical model	CSMP III, Dynamo, Midas System described as set of partial differential equations		
Data Flow Simulation	Nodes interconnected by arcs. Tokens=Data flow	DFSS simulation language	Data flow architecture and algorithm behavior	
On Line Simulation	Actual net	In situ operation for performance measurement		18
Testbed	Prototype net	Actual equipment installed in testbed environment	Investigate "real world" problems (noise, equipment failure, etc.)	18
Parallel Processors	Processor=Node	Multiple processors connected on a bus	Net emulation	19,20,21

Table 3. Network design/analysis approaches. (Continued)

However, since the solution algorithms are fairly general within their restrictions, one particular solution can be applied to a wide spectrum of models. Thus, analytic models can be designed and implemented very quickly. Also, some of their solution algorithms are computationally very efficient.

Queueing theory plays a key role in the quantitative understanding of computer-communication networks. Queues develop at each concentration point (node) in the network as messages arrive and wait for service. In fact, queueing while waiting for service is one of the primary factors in the time delay that messages encounter in traversing a network.

The actual queueing delay encountered depends on the statistics of the messages arriving and the so-called service discipline — the way in which messages are handled at each concentrator. The overall network itself can be visualized as an interactive network of queues. A queueing network consisting of two nodes is shown in Figure 9. The concentration and buffering aspects of network design, as well as routing, flow control and other overall network operating characteristics, depend critically on an understanding of queueing theory for their quantitative characterization.

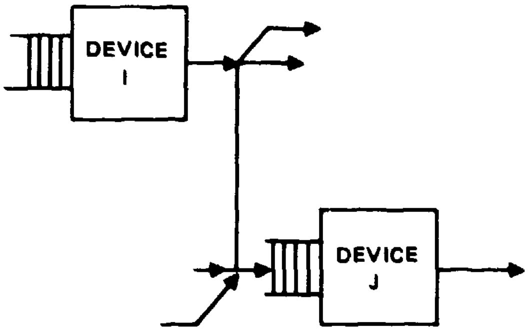


Figure 9. Network of queues.

Queueing systemɒ can be used to model processes in which customers arrive, wait their turn for service, arc serviced and then depart. Queueing systems can be characterized by five components:

1. The interarrival-time probability density function
2. The service-time probability density function
3. The number of servers
4. The queueing discipline
5. The amount of buffer space in the queues

For infinite-buffer, single-server systems using first come, first served queueing discipline, the notation A/B/m is widely used. A is the interarrival-time probability density, B, the service-time probability density and m, the number of servers. The probability densities A and B are chosen from the set

M - exponential probability density (M stands for Markov)
D - all customers have the same value (D is for deterministic)
G - general (i.e., arbitrary probability density)

The state of the art ranges from the M/M/1 system, about which everything is known, to the G/G/m system, for which no exact analytic solution is yet known.

Consider the M/M/1 queueing model of Figure 10. The M/M/1 model is often assumed when analyzing computer systems. The assumption of an exponential interarrival probability is completely reasonable for any system that has a

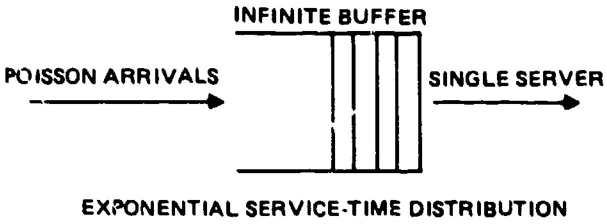


Figure 10. M/M/1 queue.

large number of independent customers. Under such conditions, the probability of exactly n customers arriving during an interval of length t is given by the Poisson law:

$$P_n(t) = \frac{(\lambda t)^n e^{-\lambda t}}{n!}$$

where λ is the mean arrival rate.

Although assuming an exponential interarrival probability density is usually reasonable, assuming exponential service time is harder to defend on general grounds. Nevertheless, M/M/1 may be an adequate approximation for situations with fewer long service times.

The state of an M/M/1 queueing system is completely described, given the number of customers currently in the system, including both queue and server. Although it would seem necessary to describe the status of the customer currently being served, the exponential density function has no memory. The probability of the remaining service time requiring T seconds is independent of how much service the customer has already received. The exponential function is the only density function with this memoryless property.

Queueing systems in which the only transitions are to adjacent states are known as birth-death systems.

Consider an arbitrary queueing system in equilibrium and let N, W and λ be the average number of customers in the system, the average time customers spend in the system and the average number of arrivals per unit time, respectively. Little's theorem states that

$$N = W * \lambda,$$

regardless of the interarrival and service time distributions, the service discipline and any dependencies within the system.

The following is one possible method for classifying analytic queueing models. This classification system is based on a model space described by the following six (reasonably independent) dimensions:

1. Model structure
2. Arrival process
3. Workload classification
4. Queueing disciplines
5. Service demand description
6. Server characteristics

The first three dimensions are global, pertaining to the entire model. The remaining three describe individual service centers. In discussing the dimensions individually, we indicate those classes that have proved to be most important in the development and the application of queueing network models. Generally, in each dimension, the classes can be ordered according to complexity, detail and cost of solution.

This classification scheme was taken from Reference 9.

The model structure describes the user of service centers and the manner in which jobs flow among them. We distinguish the following:

1. Single server model (possibly with a feedback loop)
2. Cyclic queueing model (a constant number of customers cycle among the service centers)
3. Central server model (customers move from a designated service center to other centers, but after receiving service, they return to the designated server)
4. General queueing network (arbitrary routing among service centers)
5. Hierarchical queueing network (a single service center at one level of detail is represented by a network on a lower level of detail)

The arrival process indicates the manner in which new customers come into existence. A model is one that is:

1. Closed (fixed number of customers in each routing chain)
2. Open (arrivals and departures in all routing chains)
3. Mixed (some routing chains open and some closed)

The arrival rates in open chains may be constant rate Poisson, load-dependent Poisson or non-Poisson.

The workload classes characteristic indicates groupings of customers that are statistically indistinguishable. Model possibilities are:

1. Single class
2. Multiple class with no class changes
3. Multiple class with class changes

The queueing disciplines we distinguish are:

1. Station balance (including processor sharing, preemptive last-come-first-served and no-queueing)
2. Class-independent work-conserving (including first-come-first-served)
3. Strict priority (based on customer class)
4. General

The service demand description can be specified as either

1. A workload vector, in which the mean total service required by a customer of a class at each device is stated or
2. A routing matrix indicating the pattern of stochastic movements of customers and the distribution of service times for each class at each device. The service time distribution may be assumed to have a particular form or to have specified moments.

The server characteristic describes the reaction of the server to the load. These include:

1. Load-independent servers
2. Load dependent servers

If the service rate is load dependent, it may depend on the number of customers of the same class at the service center, on the total number of customers at the service center or on the total number of customers in a subsystem. Much of the early work in networks was done by Kleinrock[6,7,8] whose approach is to model the communication network as a network of queues.

STOCHASTIC MODELS

Computer communication systems are characterized by unpredictable sequences of random demands on the available resources. The theory of stochastic process thus also provides a large and effective set of analytical tools particularly suited for the modeling of these probabilistic systems. If we carefully analyze each queueing system for which there exists some solution we undoubtedly would find an underlying Markov or semi-Markov process in a great percentage of them.

Renewal Theory

Renewal theory and the theory of regenerative processes, in general, relate to systems in which there is an underlying process which probabilistically restarts itself. The concept from renewal theory that corresponds to alternating renewal processes proves most useful. An alternating renewal process is one which describes a system that can be in one of two states, on or off. Starting in the on state, the system alternates between these two states. The periods of time it spends in each are random variables which follow a common distribution for each of the two states. The key element in such an analysis is to identify points in time at which the system regenerates itself: the interval of time separating two consecutive regenerative points is called a cycle; the ratio of average time the system spends in a given state to the average cycle time is precisely the fraction of time that the system spends in that state.[22]

Assuming an infinite population in conjunction with renewal theory arguments has allowed the determination of channel capacity under various schemes. This example (model) assumes that the traffic source consists of an infinite number of users who collectively form an independent Poisson source with an aggregate mean packet generation rate of S packets per packet transmission time T. (We assume here that each packet is of constant length requiring T seconds for transmission.) This is an approximation of a large but finite population in which each user infrequently generates packets and each packet can be successfully transmitted in a time interval much less than the average time between successive packets generated by a given user. Each user in the infinite population is assumed to have at most one packet requiring transmission at any time (including any previously blocked packet). Under equilibrium conditions, S is also the channel throughput. Two additional assumptions are also introduced: the average retransmission delay X is large compared to T and the interarrival times of the point process defined by the start times of all the packets plus retransmissions (and reschedulings) are independent and exponentially distributed.

Markov Theory

A collection of random variables $\{S(t), t>=0\}$ is said to be a 'Markov process' if the probability distribution of the state at time $t+y$ depends only on the state at time t and not on the process history prior to t.

The behavior of a Markov process can thus be described as follows: at time, $t=0$, the process starts in some state, say i. It remains there for an interval of time, distributed exponentially with parameter λ_i (average length $1/\lambda_i$). The process then enters state j with probability $q(i,j)$, remains there for an exponentially distributed interval with mean $1/\lambda_j$, enters state k with probability $q(j,k)$, etc. The successive states visited by the process form a

'Markov chain,' that is, the next state depends on the one immediately before it, but not on all the previous ones and not on the number of moves made so far. This Markov chain is said to be 'embedded' in the Markov process.

The state j of the Markov process is said to be 'reachable' from state i if there is a non-zero probability of finding the process in state j at time t, given that it started in state i. A subset Σ of process states is said to be closed if no state outside Σ is reachable from a state in Σ. Thus, if the process once enters a closed subset of states, it remains in that subset forever afterwards. A set of states is said to be 'irreducible' if no proper and non-empty subset of it is closed. As far as the long-run behavior of the process is concerned, an irreducible set of states can be treated in isolation, so we can assume that the set of all states, i.e., the Markov process, is irreducible.

Every state of an irreducible Markov process is reachable from every other state. A state j is said to be "transient" if the total average amount of time spent in state j, given that S(O)=i, is finite. Otherwise, j is "recurrent." Since the average time the process remains in state j on every visit is finite, the average number of visits to state j is finite if j is transient; it is infinite if j is recurrent.

Let us assume the Markov process is recurrent as well as irreducible. Every state is guaranteed to be visited, no matter what the initial state is. Having once visited a state, the process keeps returning to it ad infinitum. If the average length of the intervals between consecutive returns to state j is finite, then state j is said to be "recurrent non-null." If this average is infinite, then state j is said to be "recurrent null."

The moments of successive visits to state j are "regeneration points" for the Markov process.

In an irreducible Markov process, either all states are transient, or all states are recurrent null, or all states are recurrent non-null. In the first two cases, all limiting probabilities are equal to 0; steady-state does not exist. In the last case, all limiting probabilities are non-zero; steady state exists.

A semi-Markov process is a stochastic process which makes transitions from state to state in accordance with a Markov chain but in which the amount of time spent in each state before a transition occurs is random.

Markov Decision Models

In the previous Markovian models presented above, we assumed the system parameters were all fixed, time invariant and state-independent. These models are referred to as static. Clearly, it is often advantageous to design systems that dynamically adapt to time-varying input and to system state changes, thus providing improved performance. If the system is Markovian in nature, then the theory, known as Markov decision theory, provides a basis for analysis. Consider the process X(t) and its state space S labeled by the nonnegative integers. Let A be a finite set of possible actions that correspond to each action "a" chosen from the set A, a set of state transition probabilities is specified and a cost is incurred. A policy f is a rule for choosing actions. Let P be the class of all policies. An important subclass is the class of stationary policies. A stationary policy is defined to be one which chooses an action at time t depending on the state of the process at time t. It easily follows that if a stationary policy f is employed, the sequence of states {X(t), t=0,1,2,...} forms a Markov chain. It is thus called a Markov decision process and possesses stationary transition probabilities.[22]

Regression Models

A regression model may be used as a fast statistical model of computer system performance which relies on workload and performance data collected from the system being evaluated. However, it has the disadvantage of not being capable of modelling logical and structural relationships in the system. A simulation model does not suffer from this limitation. However, a simulation model which produced results similar to a regression model probably would need to model the system in considerably more detail, and consequently be more expensive to implement.

By combining simulation and regression techniques within a hybrid model, the advantages of both techniques may be exploited. Simulation modelling techniques are used to model in considerable detail those aspects of the system of particular interest. Regression modelling techniques are used to model the rest of the system in much less detail.

This information was taken from Reference 23.

GRAPH THEORY

For our purposes the computer network can be modeled by a graph G = (N,L) where N is the set of nodes of G representing the network switching centers (NSC) to which subscriber computers (SC) and terminals are connected; L is the set of arcs of G representing the communication links. If graph G contains n nodes, it can be completely described by the n by n matrix A whose i, jth element equals one, if there is a link, and equals zero, if there is no link. A is known as the connection matrix. It is often convenient to assign each link of a graph a real number called its weight. For example, the weight may be the length of the link, its cost, its transmission capacity or other useful quantity. The weight of a cycle or route is the sum of the weights of its link.

If we assign to each node a real number, we have a set of node weights which may represent cost, switching capacity and so forth.

In the design of networks for the connection of terminals to computers, the following problem arises: given a fixed set of nodes such as the cities in which the terminals and computer are located, find a tree on these nodes that minimizes total link (communications line) costs, perhaps subject to some constraints. Since the number of trees on n nodes is n**(n-2) it is impractical, except for small n, to generate all possible trees and find the cost of each as well as test it for compliance with any constraints. There is a basic procedure for finding a minimum spanning tree (MST), that is, one that has minimum weight but is not subject to any constraints. This algorithm is known as the Prim algorithm (minimum spanning tree). A minimum spanning tree is shown in Figure 11. Next, procedures can be obtained for finding trees with minimum-weight properties when there are constraints. A useful algorithm for this is the Esau-Williams algorithm. An exact solution can be obtained using the branch-and-bound method. Descriptions of these algorithms can be found in Reference 24.

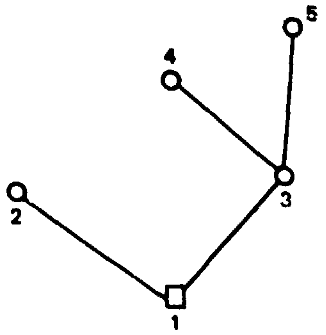


Figure 11. Minimum spanning tree.

The major practical problem relating to routes in communications networks is to find one or more routes between a specified pair of nodes, or between all pairs of nodes, subject to certain constraints. Algorithms to generate routes are desirable because, somewhat as in the case of trees, the number of routes. disregarding constraints, grows very rapidly with the number of nodes. Therefore, it is impractical to enumerate all routes and select those that are acceptable. For any given set of nodes there are a great many ways to connect these nodes. Examples of possible connections are given in Figure 12.

Floyd's algorithm addresses most directly two questions:

1. How long is the shortest route between node i and node j, for all pairs i, j with i not equal to j?

2. What links make up the shortest route between i and j? The Dijkstra algorithm addresses the problem of finding the lengths of the shortest routes from a specified node to all other nodes. It is often desirable to be able to generate routes longer than the shortest route; for example, as alternate

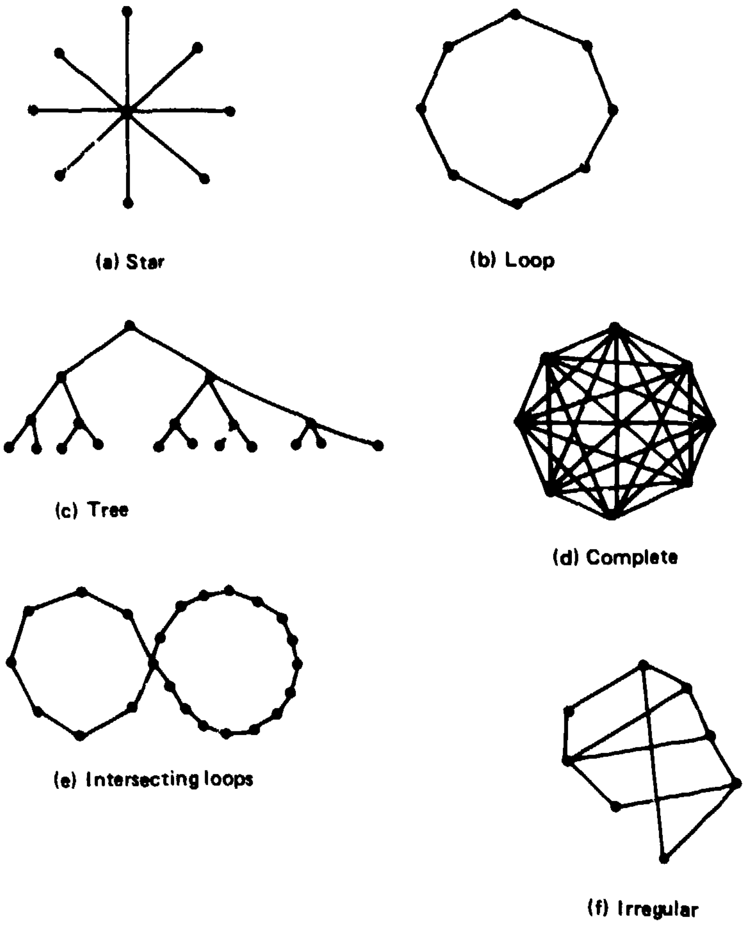


Figure 12. Possible ways to connect nodes.

routes when the shortest route is unavailable for some reason. In many practical problems, rather than identify the kth shortest routes, it is desirable to find all routes between nodes i and j whose length does not exceed M. An algorithm known as the route-tracing algorithm does this. These procedures can be found in Reference 25.

Designers often need to be able to compute the information carrying capacity of a network. The cut is a concept from graph theory useful for modeling the carrying capacity of a network. An X-Y cut is a set of arcs whose removal disconnects node X from node Y. The number cf arcs in a cut may vary from one to all the arcs in the graph. A minimal cut is one in which replacement of any of its members reconnects the graph. In a weighted graph, each cut has a capacity. The capacity of a cut is the sum of the weights of the arcs in the cut.

A variety of measures of network reliability, that is, of a network's ability to continue to provide communications routes between some nodes when other nodes or links fail, has been proposed. These measures fall into two classes: Deterministic measures depend only on the structure of the network, that is, on the numbers of nodes and links and the way they are connected. Probabilistic measures of availability, on the other hand, depend not only on the structure, but also on the probabilities of failure of nodes and links. One measure of network reliability is the cohesion of G. This is the minimum number of links that must be removed from G to break all routes between at least one pair of nodes. Another measure of reliability is the connectivity. The connectivity of a graph G is the minimum over all node pairs of the minimum number of nodes of G that must be removed from G to break all routes between i and j. The basic algorithm for finding the cohesion is an application of the max-flow min-cut theorem. To determine the connectivity, the problem can be converted into a cohesion problem and then solved.

The selection of the appropriate type of topological structure for a network depends on the application. Two different factors mainly influence this choice: reliability, which would favor highly connected topologies and total cost which favors tree-like topologies.

Techniques for reliability analysis based on graph theory are given in Reference 10.

Net Models

Another approach that can be used to model and analyze communications networks is that of net models. These net models include such models as Petri nets (developed at MIT), evaluation nets, and the graph model of behavior, GMB (developed at UCLA).

To accomplish network modeling concepts, a suitable description method needs to be found which offers capabilities of representing:

- Parallel processes and their synchronization
- The logic structure of a system part

- Transient entities and their information content
- Propagation delay or execution time.

Additionally, a graphical representation of the model structure for better comprehension and a suitability of the model description to easy implementation for simulation are required.

On approach that satisfies these requirements is evaluation nets. Evaluation nets allow the description of parallel processes, control and data flow and processing time. They offer a graphical representation and a complementary formal description suitable for simulation input. They allow hierarchical levels of description in that a given portion of a system may be expanded or compressed in detail compared to the rest of the model.

Transitions, locations and tokens are the elements of an E-net. Tokens may be regarded as information carriers. A token may represent a job, message, command or status information moving through the system. Tokens reside in locations, one at a time, and control their motion mutually at transitions. Transitions may represent time-less actions (events) or time-consuming actions (activities). There are five basic transition types shown in Figure 13. They are T-transitions, J-transitions (join), F-transitions (fork), X-transitions (output selection) and Y-transitions (input selections). A transition "fires" if the activities represented by that transition are executed. That is, after elapse of the transition time, tokens are moved from their input locations to the output locations and their attributes, representing the information carried, are possibly changed according to the transition procedure. Every transition is described by its schema, a transition procedure which affects the attributes, a time procedure if the transition time is not zero and, in the case of the X- and Y-transition, a resolution procedure which allows the control of token advance according to the status of the net or the actual values of token attributes. This information was taken from Reference 26.

Petri nets are another approach to network modeling. Petri nets are graphs representing interacting processes contending for resources and consist of transitions and places, interconnected by edges. Transitions model processing while places model the state of the system. The dynamics of Petri nets

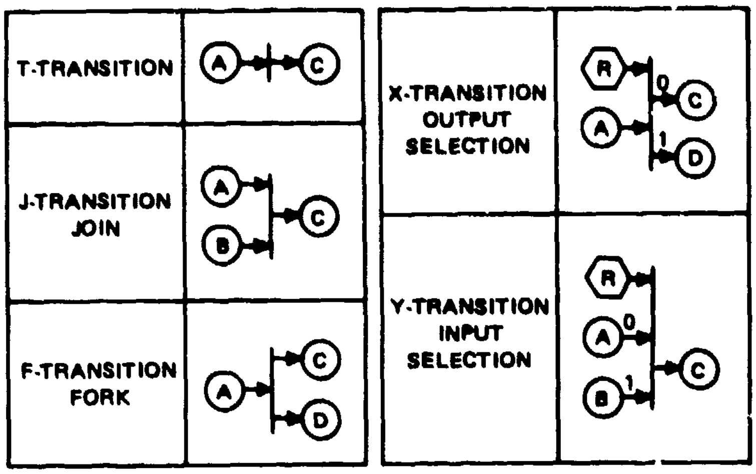


Figure 13. Basic transition types of E-nets.

are described by tokens flowing from place to place across transitions. The presence of tokens on places causes transitions to fire, thereby causing some tokens to be removed from the input places of the transitions and tokens to be placed in the output places of the transitions. The primary issue in Petri net research has been Liveness and Safeness. These two properties deal with deadlocks, traps and other control-flow anomalies. Little attention has been paid to timing issues. The Petri net simulator is APL based and provides little simulation support aside from APL's debugging aids. An example of a Petri net is given in Figure 14.

Another approach to network modeling using graph theory is the GMB. The Graph Model of Behavior (GMB) was developed at UCLA at the same time as Petri nets. The GMB is particularly well-suited for use in a structured, multi-level design environment because of the attention given to its use for both analysis and simulation. It is used as the basis for the SARA design package. A more complete description of the GMB can be found in References 27, 28, 29.

Another approach to network simulation based on graph theory is described in Reference 30. Here the execution of the simulation is not controlled by a program on a general-purpose computer. Instead, each node of the graphical description is simulated by an electronic circuit and the nodes are physically connected by means of electric wires. Conditions expressing the routing of entities through the network are also expressed physically by Boolean values issued from electronic circuits or from a computer and carried by electric

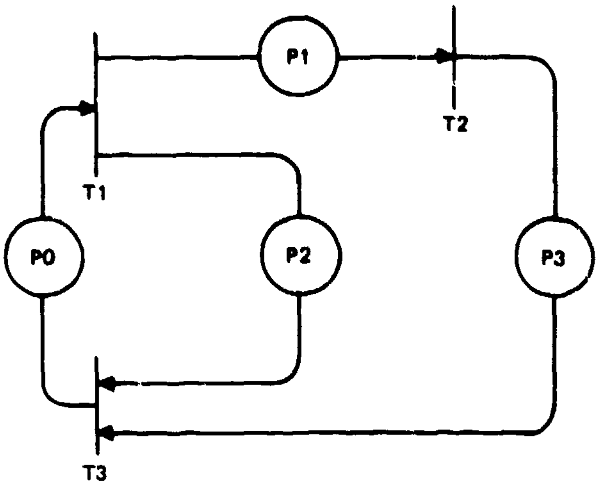


Figure 14. Petri net.

wires. The operation of this type of simulator is very similar to the operation of an analog simulator except the circuits carry binary information and enable us to perform discrete-event simulation.

FINITE STATE DESCRIPTIONS

Discrete-event simulation represents the state of the system of interest by elements of a finite-state machine (e.g., a digital computer) and observes (or measures) the state changes specified by a number of fixed rules (e.g., a computer program). The state changes usually are referenced with respect to time represented either by some fraction of real (or execution) time or by one particular variable of the finite-state machine. The time used to reference the occurrence time of state changes (or events) is called "simulation time."

Finite state models may also be used for specification and validation of communication protocols.[11] This description method basically subdivides the system into a number of communicating components, so that each component is a finite state machine. Figure 15 gives an example of using a finite state model to model a simple transmission medium.

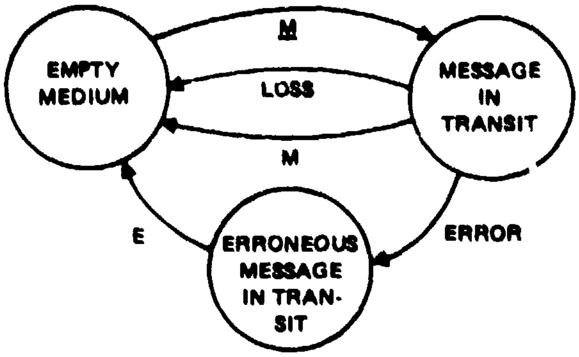


Figure 15. Finite state model of a simple transmission medium.

APPROXIMATION TECHNIQUES

Many important practical cases of large-scale computer systems are too complex to be represented exactly by a mathematical model. Even when a precise mathematical model can be constructed, the analyst is faced with a problem of dimension. Models with a number of states proportional to 1,000,000 are easy to obtain, but program packages capable of solving Markov chains of this dimension are not yet available. Often the mathematical models which arise from computer systems have properties which make them particularly difficult to handle numerically. For instance, the time constants related to various parts of the system will vary widely leading to "stiff" systems of equations. Such properties also make simulation modeling particularly difficult. If a system is composed of parts with very small and very large time constants, it will be necessary to simulate it at the time scale which corresponds to the rapidly varying portions to preserve the desired accuracy. However, the total simulation time will have to be large compared to the slowly varying parts for the simulation to reach steady-state. Furthermore, the probabilistic or statistical tools available at present do not permit us to estimate accurately the confidence intervals of simulation results, except for the simplest models which have a regenerative structure or other simplifying properties.

While the analysis of networks of queues seems to have matured in the last few years, much remains to be done for the models to reflect a more accurate picture of a data network. For example, computer queueing structures have highly irregular connections as compared to those of a data network.

Protocols for flow control play an important part in data networks; there is no general method to model such protocols. Further, there is much more coupling and correlation between queues in a communication network since short packets remain short packets as they traverse the network, etc. Some researchers reported that trying to fit a queueing model to a real problem is too restrictive and, in fact, may be misleading. Nevertheless, analysis should bring insight to permit effective use of queueing models for the analysis of data networks.[31]

All these considerations make it particularly desirable to have computationally tractable and relatively accurate approximate mathematical models for computer systems.

Diffusion Equations

One approach to approximating solutions to network problems is to use diffusion approximations which are computationally tractable and relatively accurate approximate mathematical models for computer systems. If they are used carefully, under relatively heavy load conditions and when traffic and service times do not have excessively high coefficients of variation, their accuracy is comparable to that of simulation models. The computational effort involved in solving them is negligible by comparison. Usually it will involve solving a system of linear equations whose size is the product of the number of stations and the number of customer classes and computing moments from a continuous or discrete distribution function.

The open problems in this area are of both a mathematical and a practical nature. The convergence of the queueing models to the diffusion approximations has been established only for the simplest and the least interesting cases. This is hardly surprising since the mathematical tools for this are still rudimentary. From a more practical point of view we need to further our understanding of "good" diffusion models for various cases of interest, such as queue-dependent arrival or service times, which are not yet properly handled. Also, further practical and theoretical understanding of the properties of the flow of customers in a queueing network will improve the accuracy of diffusion approximations.

A promising method for the approximation of queueing systems with general service time distributions is to use a diffusion process to approximate the number of the queue. The method would replace the discrete number of jobs in the queue by a continuous variable, which, according to the central limit theorem, will be approximately normally distributed under heavy traffic conditions. Consider, for instance, the G/G/1 queue; the basic assumption to the diffusion approximation for this model is that as soon as a busy period begins, the stochastic process representing the number in the queue is adequately approximated by the predictions of the central limit theorem.

Several questions arise in the choice of the approximate process model:

1. The choice of the appropriate boundary conditions
2. The choice of the diffusion parameters b, α which characterize the drift and instantaneous variance of the process
3. The selection of interval size which may be used to work back to a discrete probability distribution from the continuous density of the diffusion process

This information comes from References 32, 33.

The principal idea is to convert the basically discrete (and difficult) queueing problem into a continuous (and simpler) system by a limiting operation. The resulting partial differential equations resemble diffusicn equations and the entire machinery of parabolic equations can be b~ought to bear on the model. However, with this method, boundary conditions must be handled in a reasonable way. Improved schemes for handling boundary conditions result in significant improvements in the approximation. Generally speaking, the greater the fraction of time that the system spends on the "boundary," the less accurate the approximation. For instance, boundary conditions would be very important in a closed network where each customer has distinct behavior and is thought of as belonging to a different class. Similar problems arise with priority disciplines. Boundary conditions would be less important in a heavily loaded single server queue with a potentially infinite number of customers since the system would spend relatively little time at the boundary (that is, the ideal queue). This is frequently referred to as the "fluid approximation." To consider second moments it is necessary to postulate Gauss-Markov process. As a rule of thumb based on experience, if both service time and interarrival time distributions are not highly skewed and if the system is heavily loaded, then the diffusion approximation tends to be accurate. Unfortunately, however, many distributions occurring in data or computer communications tend to be skewed. The diffusion approximation is analogous to the central limit theorem in that we must be careful in applying it where the limiting operations are not justified. A very important open problem in the diffusion approximation method is to develop a technique to estimate the accuracy of the result obtained by this method.[31]

Decomposition Method

In this section we discuss approximations which retain the discrete nature of the model. The set of numerical solution procedures presented has been designed for the approximate analysis of closed networks of queues. All the procedures call upon the concept of an isolated subsystem composed of one or more queues in the network. This subsystem is examined in detail under the effect of the rest of the system viewed as an aggregate. The stationary solution will then be framed either in terms of marginal distributions for each subsystem or as a product of the marginal distributions when the stationary distribution for global network state is desired. An example of this is given in Figure 16.

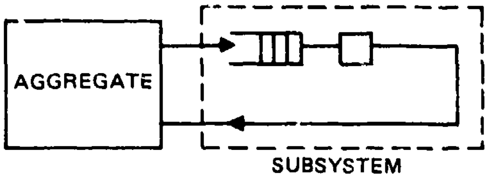


Figure 16. Simple aggregate/subsystem decomposition of a queueing network.

The basic idea is to isolate those events happening at a fast rate from the slower events. These fast event activities are treated as a subsystem and are analyzed individually so as to focus on the major events. This method is able to identify important parameters and to provide further insight into network performances. However, it is not always clear how a complex system should be optimally decomposed and the error caused by the decomposition cannot be readily computed.[31]

A technique which is closely related to decomposition was inspired by electric network equivalents. In electric networks, a complex portion of the network can be simplified by replacing it with an equivalent current source and parallel impedance, or a voltage source and series impedance. These equivalent circuits are exact equivalences for electric systems and suggest a heuristic equivalence for queueing networks.

This information can be found in Reference 32.

Iterative Approximations

This is a heuristic attempt to analyze complex closed queueing networks. Ideally the behavior of each queue in the network can be determined by analyzing that queue and its interface with the rest of the network. If the interfaces can be described in simple terms, then the problem of analyzing a system of a queue and its interface is tractable, whereas the analysis of the entire network is not. In practice, in most cases the interface is actually very complex. This attempt is to describe complex interfaces by means of simple models. The method is based upon invariants common to all queueing models. For instance, at equilibrium, the rate at which customers depart from a queue must equal the rate at which they enter the queue. Each queue coupled with its interface is analyzed to obtain queue statistics, including throughput. If the queue statistics satisfy the invariants, the algorithm stops. Otherwise the interface for each queue is adjusted in a heuristic manner and a new iteration is begun.

Unfortunately, there is at least one situation in which the method converges extremely slowly or does not appear to converge at all. This is the case of a network with a queue using a preemptive discipline where high priority customers are several times slower than low priority customers. Though this case is unlikely to occur in modeling practice, it highlights the problem with this method: this attempt cannot be guaranteed to converge at a point near the true value except in the uninteresting case where the network satisfies local balance (in which case the network satisfies product form and queue statistics can be readily computed). In most cases, empirical studies show that the heuristic procedure works well.

The algorithm is based on analogy between queueing networks and electrical circuits with throughput being analogous to current. An analogy to Norton's theorem on electrical circuits is used to simplify the given network. Figure 17 shows Norton's equivalent principle applied to a queueing network. Similar ideas have been used in non-iterative methods.

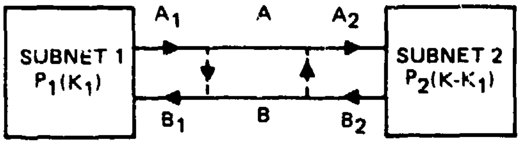


Figure 17. Norton's equivalent applied to a queueing network

This information was taken from Reference 31.

Sparse Matrix

This method permits the analytic solution of fairly large, discrete systems (up to 10,000 states) by constructing a state transition matrix and utilizing a data structure which capitalized on the regularity and sparsity of typical systems. The rate of convergence to a solution for queue lengths, stationary state probabilities, etc., can be made negative exponential in time rather than reciprocal in time. A recursive queue analyzer has been constructed which will solve a 10,000 state system in about 10 minutes of CPU time. The technique is attractive because it gives virtually exact answers including state probabilities (not just averages). However, there may be problems with ill-conditioned matrices and 10,000 states may not be large enough since the number of states grows exponentially with the number of queues in a network of queues. In particular, data networks have extremely large numbers of states.[31]

Bounds

Instead of attempting to find approximate or exact solutions, frequently it is sufficient and often more desirable to find bounds on solutions. Various workers have found exponentially tight bounds on M/G/1 and G/G/1 queues. A relatively simple Chernoff-type bound in analytical form can provide great insight to design. Although bounding techniques have been developed for simple queues, virtually nothing has been done for networks that would be required to model data networks.

HYBRID SCHEMES

The complexity of all but the simplest communication networks makes the development and analysis of the networks very difficult. Two extremes of models are available — analytical models and simulation models. The analytical models are limited in the amount of detail they may represent and restricted by their computational solution algorithms. Their advantages are that they can be designed and implemented very quickly. If desired, very complex systems can be characterized by a small number of parameters if certain simplifying assumptions are made. Simulation models, on the other hand, can model very detailed networks but at the cost of more time to model and implement and more computation time. Simulation models should be used when a model violates the basic assumptions of the analytical model or when models need to be extremely accurate. Often analytic and simulation techniques can be combined in a hybrid approach to network modeling. In the hybrid approach we can combine the advantages of analytical techniques with simulation while minimizing the disadvantages of both techniques.

One such hybrid model has been proposed for the analysis and synthesis of the EHF network. This model is described in Reference 34. Rubin's emphasis is on using analytical techniques whenever possible in conjunction with simulation techniques when they are required.

Another hybrid model is discussed in Reference 23. By combining simulation and regression techniques within a hybrid model, the advantages of both techniques may be exploited. Simulation modeling techniques are used to model those aspects of a particular system in considerable detail. Regression modeling techniques are used to model the rest of the system in much less detail.

This information was taken from Reference 23.

MODULAR SCHEMES

A modular or structural simulation model is a method employing the following characteristics:

1. Divides subnetwork activities into isolated, nonoverlapping sets of functions, each with its own rules, which we shall call protocol areas.
2. Provides subnetwork structure initially consisting of the names of the computer subprograms, each of which will define a protocol area; these subprograms (supplied by user or standard library) are the building blocks available for constructing models of specific networks. Later, the executive program will retrieve these subprograms by name and use them in constructing the model.
3. Sets parameters of the network to provide those data values consistent with and needed by the model.

This type of model gives the user two distinct levels of control:

1. The ability to modify any of the model's parameters to do simple parametric experimentation.
2. The ability to replace subprogram currently implementing the rules controlling a protocol area with another subprogram.

Additional information describing the VANS (Value Added Network Simulator) is given in the following two tables. Table 4 gives the input parameters

Parameter	Type	Meaning
N	Scalar	Number of nodes
MAXHOST	Scalar	Maximum number of hosts at each node
STARTTIME	Scalar	Time to begin collecting statistics
SIMTIME	Scalar	Total simulation time
PRIORITY	Scalar	Number of static priority levels
OVERHEAD	Scalar	Message overhead
WAIT	Scalar	Reassembly time out period
PACKETSIZE	Scalar	Size of a message packet
TURN	Scalar	Half duplex turnaround time
MAXLENGTH	Scalar	Upper bound on message length
TI	Scalar	Time increment for routing control
CMP	Scalar	Control message priority
SEED	Scalar	Random number speed
REPORT-LEVEL	Scalar	Quantity of output desired
HOST	N x MAXHOST x 7	Host descriptors for each host
CI	N x 7	The 7 CI descriptors for each CI
LINE-SPEED	N x N	Line speed/topology table
LINE-TYPE	N x N	Line type table
LINE-FAILURE	N x N	Line failure table
LINE-LENGTH	N x N	Line length table
ROUTE-TABLE	N x N	Routing table
COST-TABLE	N x N	Cost table
DESTINATION PROBABILITY-TABLE	N x N	Used to determine destination of newly generated messages

Table 4. Parametric input to the VANS system.

to the VANS system including the type of variable and what each parameter represents. Table 5 tabulates another set of parameters controlled by VANS. Here, the parameters are indexed for easy reference in the simulation. Their functions are described and their role is defined. This information was obtained from References 12, 13, 14.

The hierarchical approach appears in the following procedure. A gross or macro model is used to describe the total system; major hardware subsystems are represented as single (rather than multiple) resources in the macro model. Typically, a major subsystem may be represented by a queue and a single server in a queueing network macro model. The parameters of the single resource representation of a major subsystem are determined by analysis of a micro model for that subsystem. For instance, if a subsystem is represented by a single server in a queueing network then service rates would be determined from a micro model. A micro model is a detailed model of a subsystem. (However, portions of the micro model could be expanded into even finer submodels.)

There are several benefits of hierarchical and structured modeling and obviously they are similar to the benefits of structured programming and design:

1. The models being analyzed can be kept to tractable size
2. The model structure can be tailored to the level of detail available for parameters and validating
3. The component and subsystem models can be validated separately from the total model
4. The component subsystem can be conveniently represented at several levels of detail as is appropriate
5. System models with a very high (implicit) degree of complexity can still be analytically (though approximately) solved. This renders extensive parameter analysis of models economically and computationally possible.

This information was obtained from Reference 15.

Index	Name	Node process	Function
1	Destination	HOST	Determines message destinations
2	Length	HOST	Determines message lengths
3	Priority	HOST	Determines static priority levels
4	HOST-FLOW-control	HOST	Can turn on or off the means of communication between HOST and CI
5	Intermessage-time	HOST	Determines time of next message generation
6	Enqueue	HOST	Enters messages into HOST queue
7	Dequeue	HOST	Removes messages from HOST queue
8	Travel-time	HOST	Determines delay along HOST/CI link
9	CI-Queue-builder	BUFFER MANAGER	Adds newly arrived messages to the unprocessed message queue
10	Garbage-collector	BUFFER MANAGER	Performs housekeeping functions on all queues in the CI buffer
11	Preallocated	BUFFER MANAGER	Implements the operations involved in preallocation of CI buffer storage
12	CI-Flow-control	BUFFER MANAGER	Can turn on or off the means of communications between HOST and CI
13	CI-Queue-selector	CI	Selects message from the unprocessed message queue for processing by a CI
14	Priority-changer	CI	May temporarily alter the static priority of a message as it progresses through the system
15	Disassemble	CI	Converts a logical message (that seen by user) to physical message (message to be transmitted to next node)
16	Error-detector	CI	Determines whether newly arrived physical message is correct

Table 5. Parametric input to the VANS system.

Index	Name	Node process	Function
17	Error-corrector	CI	Implements the recovery protocols for arriving messages containing errors
18	Create-control-message	CI	Creates all subnetwork control messages required by protocol
19	Process-control-message	CI	Routes all control messages to the proper protocol module
20	Store-and-forward	CI	Manages the store/forward queue
21	Router	CI	Handles the routing of messages through the subnetwork and modifying the routing and cost tables
22	Shortest-path	CI	Determines the all-pairs shortest path through the cost table
23	Reassemble	CI	Converts a physical message back into a logical message
24	End-to-end	CI	Implements any necessary end-to-end CI protocols
25	LI-Queue-builder	LI	Manages the line interface queues
26	LI-Queue-selector	LI	Determines the order in which messages will be given to the line interface computer
27	Transmit	LI	Determines the transmission time of a message across a physical link
28	Dial-up	LI	Models the delay involved in creating a circuit-switched connection protocol
29	Enable-to-send	LI	Determines if we are able to transmit a message across a physical link
30	Half-duplex arbitrator	LI	Allocates a half duplex line between two competing nodes
31	Backing-delay	STORE	Models the seek, latency and transfer delays associated with the backing store

Table 5. Parametric Input to the VANS system. (Continued)

Index	Name	Node process	Function
32	CI-fail	GREMLIN	Determines probability of CI failure
33	Line-fail	GREMLIN	Determines probability of the total failure of a physical link
34	Error-handler	All	General error handler for all node processes
35	Scheduler	All	General event scheduler for all node processes

Table 5. Parametric Input to the VANS system. (Continued)

DISCRETE-EVENT SIMULATION (USING SIMULATION LANGUAGE)

Special purpose simulation languages can be used to simulate communications networks. General simulation languages differ from general programming language in that they have some extra features needed in simulation, such as queue handling, timing functions, transaction processing, priority functions, etc. With simulation language it is possible to construct any kind of model as accurately as desired.[6] Simulation models can model very detailed networks but they take more time to model and implement and require more computational time. In general, this approach to network design is most appropriate when the problem is too difficult for closed form analysis but is an isolated and well-described problem.

Many papers dealing with computer communication networks tend to focus on one problem area. They are characterized by a fixed choice of higher-level rules that determine network behavior. The user's only control over these models is choosing the values of certain network parameters, such as the number of nodes, line speeds, message frequencies, message lengths and channel error rates. In these models higher level subnetwork tasks (such as error detection and correction, routing, acknowledgement procedures or flow control protocols) are usually not part of the user/model interface and are not under the user's control. They can be changed only by major modification to the program.

These models give good results for specific situations but have two severe limitations. They cannot be used to analyze networks with different architectures and they cannot be used for much of current research concerned with more than simple parametric investigation.

To meet the needs of a computer network analyst, the modeling language has to be a high level language suitable for describing processes typical for data communications networks. Typical characteristics of data networks are concurrent software processes inside a node computer and completely independent processes running on different machines spread around a large geographical area. The language has to be easy to read (i.e., English-like) and it must permit independent software modules to be put together to form a complete software system.

Three types of languages have been used for simulation: general-purpose programming, simulation and descriptive/design languages. Each type of language has advantages and disadvantages which should be considered by users in selecting language best suited for their applications. A tabulation of the various programming languages is given in Table 6.

Language	Language Purpose	Systems Modeled	Language Style
FORTRAN, PL/I ALGOL, APL	General purpose	Continuous and/or discrete systems	Conventional
CSMP III DYNAMO MIDAS	Simulation	Continuous systems	Conventional
GASP IV	Simulation	Continuous and/or discrete systems	Conventional
SIMSCRIPT II.5 SOL, SIMULA	Simulation	Discrete systems	Conventional
GPSS	Simulation	Discrete systems	Graph based
ISPS, MPDL AHPL, DCDL	Description/ dusign	Discrete systems	Conventional
DFSS Petri nets GMB	Description/ design	Discrete systems	Graph based

Table 6. Programming languages for network simulation.

The arguments in favor of general purpose programming languages are:

1. A larger group of people reads and writes programs written in or using general purpose programming languages. Therefore, the pool of programmers is larger and the programs are more acceptable.

2. A larger group of computer systems supports general purpose programming languages. Therefore, simulation programs written in them are more portable.

3. There is greater effort to decrease cost of executing general purpose programming languages. Therefore, they are usually cheaper to use.

The disadvantages are:

1. Each model must be coded entirely from scratch, including simulation functions which are common to many models. Therefore, only those modelers who are familiar with the general purpose programming language program will readily read and understand the models.

2. General purpose programming languages contain few simulation oriented functions and provide little simulation oriented debugging aid. Therefore, those who design and carry out simulation experiments are in a weaker position.

Descriptive/design languages are languages developed to describe the behavior of some existing systems or to design new systems. These languages generally attempt to accurately reflect the structure and behavior of the system being described or designed. Many descriptive/design languages have as primary goals both analysis and simulation. Some of these languages also deal with the difficulty of bridging the gap between modeling and implementation. Modeling artifacts often found in simulation languages are minimized in descriptive/design languages. Interspersing measurement and report generation constructs within the model, therefore, is minimized, often leading to a weakness in these areas.

Simulation languages are languages developed expressly to conduct simulation experiments. As would be expected, these languages are characterized by their strong support of basic simulation functions, in particular, instrumentation and report generations.

Most simulation languages are characterized in the literature by the types of physical systems which they can model and simulate. In particular, it is important to note whether the physical system variables (also called state of dependent variables) change smoothly or in discrete steps. Continuous systems are those whose variables change smoothly. Such systems are most often described by partial differential equations or finite difference equations. Discrete systems differ from continuous systems in that system variables change in discrete steps according to some stated algorithm.

FORTRAN, PL/I, ALGOL and APL have been used extensively to construct simulation models. FORTRAN provides efficiency, PL/I provides powerful constructs, data structures and block structure (as does ALGOL), while APL provides powerful data operators. APL offers the added advantage of an interactive environment with useful debugging facilities. The general subroutine mechanism provided in these languages (especially PL/I and ALGOL) can be used to achieve some degree of abstraction. PL/I data structures are especially useful for data abstraction.

CSMP III, DYNAMO and MIDAS allow users to solve problems by programming them directly from a mathematical model. The languages are mostly nonprocedural and allow the user to describe the system as a set of partial differential equations. A variety of integration methods are available to the user.

GASP IV is a FORTRAN-based simulation package providing basic simulation functions such as: event control, state variable updating, information storage and retrieval, initialization, data collection, program monitoring and event reporting, statistical computation, report generation and random deviate generation. In addition, GASP provides some functions which must be replaced by the user. These "stubs" are responsible for: subprogram initialization, description of equations and initial conditions for "state" variables, event code definition, definition of event processing procedures and data collecting and reporting. GASP is somewhat limited in its modeling of queues by FORTRAN's lack of list-manipulating routines and by the lack of data structures.

GPSS is the most widely used simulation language for discrete system simulation. A GPSS model consists of a set of interconnected blocks modeling transformations. Lines connecting blocks describe possible flow paths. The dynamic portion of a model consists of transactions which flow through and are transformed by blocks. The flow of transactions can be affected by parameters of transactions and by conditioned wait statements. Standard Numerical Attributes are information items made available to the GPSS user during simulation, e.g., the current value of the simulation clock, queue lengths, etc.

In GPSS, resources are modeled by facilities and storages. Facilities are resources which can be used by no more than one transaction at a time, while storages may be occupied by more than one transaction. GPSS allows some synchronization to be modeled through the use of constructs which create and merge copies of transactions. GPSS is efficient and provides many built-in statistical measurements in addition to some user controlled monitoring and measuring facilities. Report generation is strictly automatic. GPSS has a large number of different blocks, making it difficult for inexperienced users. The textual format of the language is rigid and rather obscure. On the other hand, GPSS doesn't require previous knowledge of computer languages and the graphic form of the language is more natural to inexperienced computer users than conventional computer languages. The approach used in GPSS to simulate the changes in system state is process interaction. Its underlying structure attempts to move customers to complete their activities. Transactions (or customer records) are moved from the future event chain to the current event chain at the appropriate event times. The current event chain is then scanned and the active transactions are moved as far as possible towards the completion of their activities.

SIMSCRIPT is a classic discrete event simulation language. The underlying structure of SIMSCRIPT attempts to invoke the event routines at the appropriate simulated time instance. A program in SIMSCRIPT, therefore, contains a group of subroutines which characterizes the state changes for the different types of events and a timing routine which pro⁻esses events by calling these subroutines. Models are characterized by their events. Events are divided into exogenous (triggered from the environment, i.e., data cards) and endogenous events (triggered by other events within the modeled system). Timing mechanisms are very general and simple; each event must predict the occurrence time of events it triggers. System variables are modeled by entities which are either static (permanent) or dynamic (temporary). SIMSCRIPT's strength lies in:

- An English-like syntax which is easy to use and which is capable of concisely modeling complex interrelationships
- Powerful queueing mechanisms including user controlled queueing strategies
- Strong concepts of events, entities and sets
- Powerful built-in (but under control) measurement mechanisms
- Powerful English-like report generation constructs

SIMSCRIPT is best suited for system level performance evaluation and tuning. It is weak in its modeling of low level timing. The introduction of PROCESS and RESOURCE statements suggested by Russell increases SIMSCRIPT's suitability for modeling computer systems.

SOL was developed originally by Knuth to include the essentials of GPSS but with a symbolic language syntax (based on ALGOL). As in GPSS, the dynamics of a model are described by transactions flowing through processes. Facilities and stores model resources. The TABULATE construct is used to gather statistics. Many powerful built-in statistical analysis operations are performed. Report generation is partially automatic and partially under user control.

SIMULA was first proposed by Dahl and is an ALGOL-60 based simulation language. Its distinguishing feature is the powerful "class" concept. The ability to concatenate "classes" provides SIMULA with a data abstraction mechanism which is conspicuously absent from other simulation languages. In addition, SIMULA contains new operators to handle co-routing. The simulation support of SIMULA consists of three classes:

1. SIMSET, which provides set (and queue) manipulation routines
2. SIMULATION, which supports the concepts of processes, scheduling, synchronization and time management
3. BASICIO, which provides input/output facilities which differ from the ALGOL-60 IO facilities

In addition, SIMULA provides some synchronization and interruption constructs which make it particularly well suited for modeling resource sharing and priorities. It also provides routines for variate generation. The principal measurement capability is to accumulate (integrate over time). The principal report generation facility is histogram.

The class concept in SIMULA allows the programmer to define objects where both data and operations are included. The class objects can be created dynamically and deleted during the simulation period. (For example, an information data packet is only alive as long as the packet is transported through the network.)

A class declaration can be prefixed with other user or system defined class names. If so, the prefixed class also owns the data and operations defined in the class used as prefix. Such predefined system class is SIMULATION which, when used as prefix, allows use of several convenient tools for describing concurrent processes. Parallel processes in SIMULA can be described as process classes. In addition to the time scheduling primitives, SIMULA also contains tools for synchronizing processes, like queues and procedures for treatment of queue elements.[17]

SOL and SIMULA are used strictly in a batch environment. SIMSCRIPT is usable both interactively and in a batch environment and provides some debugging aids through the use of "monitored variables." SOL and SIMULA differ from other languages in allowing recursion. The difficulties introduced by such dynamics are discussed in a later section. None of the languages deal with the difficult issue cf multilevel simulation.

ISP, AHPL, DCDL and MPDL are classified as discrete system, descriptive/ design languages.

ISP was designed to describe instruction sets of processors. Its suitability for describing systems at the register-transfer level is widely recognized. Its initial use was in describing systems for the purpose of analysis and automatic generation of assemblers, code generators, test sequences and other applications. More recently the emphasis of its use has shifted to hardware systems design. Recent work has included symbolic execution of ISP descriptions. The strength of the ISP notation is in its useful constructs for: bit manipulation, register overlaying, memory accessing, opcode decoding, logical and arithmetic operations.

Parallelism can be modeled at the statement level (using ';'). Delays can be used to model timing. The recent emphasis on design has resulted in the addition of higher level control flow constructs relating to interprocess communication. ISP is limited in its power of abstraction by its lack of data structures and the absence of structured mechanisms for interprocess communication. The wide range of applicability of ISP has resulted in some cases in the weakening of the "semantics" of the ISP constructs. The formal semantics have often been left up to the application (analysis and simulation) programs. Recent work has attempted to deal with this weakness through the use of denotational semantics. ISP is considered weak in its support of simulation.

AHPL is used to describe and design hardware. AHPL is an APL-based register transfer level language. APL's powerful data operators are not fully utilized in AHPL to achieve the goal of taking a design to implementation. The use of powerful APL operators results in creating large amounts of hardware without the designer's knowledge and, therefore, is discouraged. Parallelism is modeled using Diverge and Converge operators. Synchronization among modules is modeled using Wait and Delay operators. Data flow is realized through assignments to registers located in different modules. AHPL attempts to deal with two levels of simulation by allowing some modules to be described as Logic subroutines. AHPL has been used for simulation but is weak in its support of simulation.

DCDL was created specifically to design digital hardware or firmware using an algorithmic language to express software functions which then are candidates for design in hardware or firmware. It deals with the problem of multilevel design by allowing the designer to model using a Logic, a Microprogram and a Simulate (algorithmic) section, all integrated with a Declare (Buffer) section. FORTRAN is the algorithmic language used to model the function of sections of the system not yet designed. The Microprogram language is well suited for high level hardware and low level software; the Logic language is well suited for modeling low level hardware. The interface between the Microprogram and Logic level and the algorithmic level is dealt with explicitly in DCDL in a Buffer section. The DCDS is strictly a batch simulator. One of the unique features of the DCDS system is the "Consequential Call" mechanism defined for simulating at the logic level. This mechanism performs dynamic reevaluation of logic variables until a steady state for the logic circuit is reached. A check is made during dynamic reevaluation to determine if a steady state is reachable. If an inconsistency in the circuit is detected (no steady state can be reached) the designer is warned.

MPDL is a language based on ISP and is currently under development at ISI. The main differences between ISP and MPDL are:

1. MPDL supports interprocess communication. Each independent process can be specified independently and connected to other processes through "connectors."

2. The semantics of MPDL, and especially the level of parallelism, are better defined than those of ISP.

MPDL lacks data structures and, therefore, is limited in its power of abstraction. Timing constructs are simple and high level. Some constructs of MPDL (Timeouts) are particularly well suited for modeling communication protocols.

The Data Flow Simulation System (DFSS) is used for a variety of purposes, including research on alternative data flow architectures, data flow languages and algorithmic behavior. The DFSS is capable of simulating the different versions of data flow models. Data flow models consist of nodes interconnected by arcs. Tokens flowing from node to node across the arcs model both control and data flowing through the system being modeled. The strengths of the Data Flow Simulation Systems are:

1. Data flow models provide a convenient and clean (no side-effects) method for expressing the parallelism inherent in algorithms and designs.
2. A large set of primitive nodes is available.
3. The simulator is interactive and provides both static (tracing) and dynamic (breakpoints) debugging aids.
4. The simulator provides built-in measurement mechanism.

In addition the DFSS allows designers to abstract the behavior of data flow modules. Data abstraction is available through data structures, although the mechanism for defining data structures is not convenient. Like SOL and SIMULA, DFSS allows recursion.

This information was obtained from Reference 29.

ON-LINE SIMULATION

One solution is to use the network itself for simulations. However, the resources of an operational network are often required on a fulltime basis and, once in use, cannot be removed from service for any length of time. The type of simulation discussed here would operate in situ, providing performance estimates of new techniques while the network is functioning. Such a simulation, running as a background task to normal network operation, would gather appropriate data on some set of proposed network changes. It is clear that by using this on-line technique, the most realistic data possible can be applied to the design problem. Furthermore, by having this simulation take place on-line, very long runs can be conducted and the impact of day to day variations and network component outages can be ascertained. Since the network node can partially process the data it measures, less data need be recorded for subsequent analysis, thereby both simplifying and greatly speeding up the design task. This methodology can be applied to the implementation of a new communications protocol; neighboring nodes could be programmed to simulate a new or modified protocol in conjunction with the protocol that is currently operational.

The scheme suggested can be applied to great advantage on problems of a local nature. On the other hand, it would be difficult to apply the technique to determine the effect of more global changes on the network. For example, if, instead of changing the characteristics of a particular line, the effect of adding a satellite link between two distant nodes were contemplated, the technique would be inadequate because it would require significant cooperation between the nodes involved to conduct the on-line simulation. For this reason it is necessary to extend this technique to cover such problems. This extension is much more complicated than the original technique.

A distributed simulation, in which two or more network nodes gather data on some proposed network modification, can yield significant benefits since it will be able to determine the effect of quite compli ted changes. For example, if a new broadcast satellite link were to be included in the network, then it could be analyzed by having the nodes that would participate in the broadcast communications implemented with a subsidiary simulation program. This would determine, in a cooperative manner, the effect of a satellite link on their throughput and delay. Clearly, such a simulation is quite difficult, but it would be beneficial because it might point out some potential pitfalls of the broadcast link before its implementation.[18]

TESTBED APPROACH

An alternative to simulation is the "try and see" approach: installing the new equipment in a test bed and gaining actual operational experience with it. The main advantage of this method is that we can obtain completely realistic answers to certain questions. We can investigate the effect of real world problems such as noise, equipment failures and so on, especially over a prolonged period of time.

On the other hand, it is very difficult to obtain sufficient realism for other types of network situations, especially those involving large numbers of nodes, lines or subscribers. No test bed can be large enough to contain all of the situations of interest for test and analysis. Problems of scale (network algorithms that become inefficient at a certain size of the network) are very difficult to study either by simulation or in a test bed. Likewise the problems of interaction between different levels of protocol in the network or between the hardware and the software of the network computers are very difficult to study in a comprehensive manner in a test bed since only a small fraction of the real operating conditions can be simulated.[18]

PARALLEL PROCESSORS

A natural approach to exploiting the parallelism typically available in the simulation of queueing network models is to treat the simulation of each server as a separate component.[19,20,21,35] A packet-switching network consists of a number of geographically separated processors, called nodes, joined together with communication channels. The function of each node is to accept addressed messages on its incoming channels from subscriber machines and from other nodes of the network and select an appropriate output channel for each message according to its destination address. Each node operates in parallel with all of the others, processing messages as they arrive.

In performing a simulation of such a packet-switching network, a natural approach to structuring the simulation is to consider each node in the network as a separate subsystem of the whole system. We can then model each node as a separate process and tie these processes together as a set of cooperating parallel processes, with messages passing as the method of interprocess communication. A message sent between two processes then represents a message sent over the communications channel between the corresponding nodes. Each component is responsible for processing the events related to its server and the actions of these components are synchronized so that the simulation is carried out correctly.

This distributed approach potentially can result in a speedup of the total time to perform a given simulation if a network of processors is available. Economical developments of such networks are becoming more and more viable with the availability of low-cost processors. An upper bound on the parallelism available is given by the number of processes into which the simulated system can be decomposed.

Decomposition Into Components

One technique to perform simulation in a distributed manner is to decompose the simulation into components and develop a method to synchronize the action of these components. Each component or node must maintain its own clock (time). Also, each processor in a distributed system has access only to its own local memory. This approach makes shared variables rather difficult to implement but, in doing so, prevents memory contention problems and encourages message passing for interprocess communication. There is no fixed rule to do the decomposition, although a natural approach is to treat each server in the network model as a separate component. However, for the case of a packet-switched network where a number of switching computers are connected together by a collection of communication channels, it may be more convenient to treat each switching computer and its outgoing channel as a component.

Interconnection Graph

The movement of customers in the network model is represented by events sent among the various components. The function of each component is then to receive customer arrival events, process them, generate customer departure events and send these events to other components. It also performs the necessary synchronization functions and collects statistical data for output purposes.

In event driven simulation, the simulation time sequence is monotonically nondecreasing, but is not an arithmetic sequence. The sequence values represent times at which the state of the system changes. We call such state changes events.

Since each component is a producer and consumer of events, we find it convenient to characterize the interrelationship of the components by an interconnection graph. Each component in the simulation is represented by a node in this graph. If component i generates events to be consumed by component j, then there is a link from node i to node j. Input links are for the reception of customer arrival events, while output links are for the departure of customers to other components. It follows that nodes must generate events for their output links in nondecreasing simulation time order. In an open network a component which generates external arrivals is represented by a node with no input links. Similarly, the sink which absorbs all departures from the network is represented by a node with no output links.

Synchronization Of Components

For the purpose of synchronization, we assume that the components communicate with each other by message passing. Since events are processed in nondecreasing simulation times, before a component accepts an arrival event, it must ensure that no other arrivals can occur at an earlier time. The acceptance of the next arrival event from one of the input links forms the core of the synchronization function. For the case of a single input link, the synchronization is trivial. The situation becomes more complicated when there is more than one input link. This component must then perform a merging function to determine the arrival with the earliest event time across all the input links. The operation is still quite straightforward when there is one or more arrivals on each input link. In some cases, however, there may not be an arrival on one of the input links for some period of time. During this period, the component cannot proceed with its part of the simulation and the amount of parallelism is reduced. When there is no customer across an input link, we say the link is "empty."

Link Time Solution

One technique to improve the amount of parallelism is to have the components send timing information to each other. This gives the greatest individual autonomy to each processor by allowing a server to step its simulation time forward to the time of the earliest event at that server.

We now characterize the merging operation by defining a link time across each link in the interconnection graph. When the link is non-empty, the link time is simply the arrival time of the next customer on that link. When the link is empty, the source end is responsible for maintaining the link time as the greatest lower bound on the next departure across the link, according to the information it possesses. In performing the merging operation, a node first determines the input link with the lowest link time. If the input link is non-empty, then it accepts the next arrival over this link, otherwise it is blocked. A blocked node becomes unblocked only when the source end can calculate a sharper lower bound or generate a customer across the empty link.

Unfortunately, with certain combinations of network topology and customer placement, a node may not be able to do either of the above and the simulation halts in a deadlock situation. The following theorem gives the necessary and sufficient conditions for a deadlock to occur:

Theorem: A deadlock exists if, and only if, there is a cycle consisting of empty links which all have the same link time and of nodes which are blocked because of these links.

This theorem gives us the means to find a solution to the deadlock problem. We first observe that acyclic graphs present no problem since the necessary condition that a cycle exists is not fulfilled. Also, if we insist that each customer must not have zero service time at any node, we can break the necessary condition that the link times around the cycle of empty links are identical.

Although deadlock can be avoided by insisting on a minimum service time at every server, the presence of cycles may cause severe degradation in the efficiency of the distributed approach.

Experimental Results

The results obtained for some very simp.:a networks are very encouraging, since the time to completion steadily decreases as the number of processors goes up. However, enthusiasm should be tempered by realizing the particular topology used is probably the best possible configuration for the distributed approach to be successful.

Conclusions

The distributed approach to discrete simulation has been evaluated by experiments performed on a network of microcomputers. It was found that for some topologies of queueing network models, this approach results in a speedup in the total time to complete a given simulation. However, for othe.: topologies, especially those with loops, the speedup may not be significant. It was also observed that the amount of message passing has a significant affect on the performance of the distributed approach.

CHAPTER 3

TOPOLOGICAL DESIGN AND OPTIMIZATION TECHNIQUES

The basic problem in the topological optimization of computer networks is to specify the location and capacity of each communication link within the network. The design objective is to provide a low cost network which satisfies constraints on response time, throughput, reliability and other parameters. It turns out that the full problem is completely intractable. Even the largest computers in the world cannot optimize a 50-node network. However, useful approximations have been developed for various subproblems. The approach is to take one subproblem at a time, isolate it from the main problem, develop algorithms for its solution and then combine the results into the solution of the overall network design problem.

In this chapter we present the network design problem and some suggested solutions. First, some background on the problem is given. Already, we see what a large, complex problem this is. Next, we define four optimization problems and discuss methods of solving them. Then, we discuss some of the differences in designing centralized versus distributed networks.

TOPOLOGY DESIGN PROBLEM

Currently, the design techniques in use do not look at this most general problem, but begin with assumptions about certain portions of the network regarding topology and method of operation. Several particular structures have been analyzed with substantial success. Heuristic topological optimization procedures for these structures have been developed and applied to practical network problems.

1. Centralized network problems require layout of lines and location and characteristics of multiplexers, concentrators and multidrop lines. Generally, good methods developed for this problem are believed to give solutions within 5 percent of optimum.

2. Loop networks are similar to the classic traveling salesman problem which has been extensively studied and for which efficient algorithms are available for moderate sized problems. As the number of points grow, subloops have to be introduced to make the problem tractable.

3. Distributed structures which connect backbone message processing nodes have been extensively treated for systems such as ARPANET and AUTODIN II. Design techniques incorporate fixed (rather than dynamic) routing and analysis procedures and, with this simplification, can produce effective designs with on the order of 100 backbone nodes.

4. Hierarchical designs are common for larger networks and different levels may be treated by different methods. In particular, certain levels may be designed as centralized or loop structures. Design procedures currently in use require extensive man-machine interactions to define the hierarchy, select the local access techniques and partition the problem into manageable subproblems that can be treated via the procedures for handling items 1 through 3 above. Specific areas which still need attention are to determine the number of levels (in fact, even defining what is a level!) and avoid predetermining what different levels will look like.

The hosts and terminals are the ultimate producers and consumers of information. For most design purposes hosts and terminals are equivalent and frequently it is convenient to lump them together under the term location or site. The traffic matrix tells how many packets per second on the average must be sent from site i to site j. When a new network is being designed, the traffic matrix is often unknown. In this case, it is common to assume it is proportional to the product of the populations of the two sites, divided by the distance between them. The probability density function for packet length is assumed known and the same for all sites.

The cost matrix tells how much various speed (leased) lines from location i to location j cost per month. The cost matrix covers IMP and concentrator locations as well as customer sites. In general, the cost of a line depends on distance and speed in a highly nonlinear fashion. There is also a fixed charge (e.g., modem depreciation) that depends only on the speed, but not the distance.

Just to complicate things further, only a discrete set of speeds is available. The optimum speed may not be a member of this set. In this case, the affect of rounding off the line speeds may produce a configuration that is far from optimal. Performing the optimization using only the allowed set of speeds is a difficult integer programming problem.

Network designers are expected to meet prespecified goals. Two common performance requirements for networks are reliability and throughput. A reliable network will not collapse if one IMP or one line goes down.

The parameters that the network designers can adjust to achieve their goals are the topology, line capacities and flow assignment (routing algorithm). The object of the entire exercise is to produce a minimum cost design that meets all the requirements. Occasionally the problem may be turned around. There is a certain budget available and the goal is to minimize the delay subject to the available amount of money.

Any way you look at it, the problem is huge.

The basic principle upon which networks are based is that the customers are not spread around uniformly. They tend to be found mostly in cities and not so much in rural areas. Consequently, a hierarchical strategy seems like a reasonable approach for computer networks. A cost effective structure for a large network is a multilevel hierarchy consisting of a backbone network and a family of local access networks. The backbone network is generally a distributed network, while the local access networks are typically centralized systems. In special cases, the network may consist primarily of either centralized or distributed portions. The topology design problem can then be broken down into several subproblems: (See Figure 18)

Backbone Design

1. Backbone topology
2. Line capacities assignment
3. Flow assignment

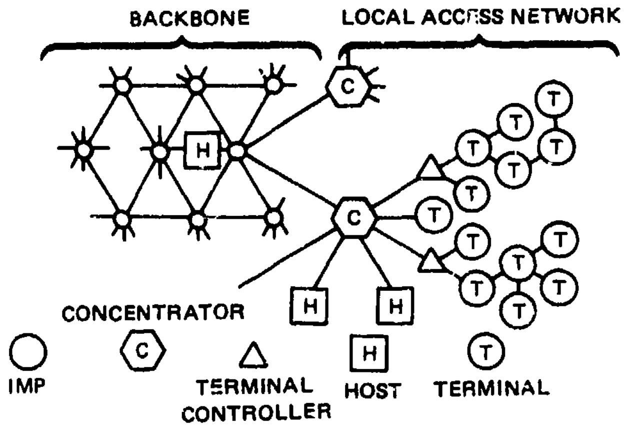


Figure 18. A two level hierarchy: backbone and local access network.

Local Access Design

1. Where to place the concentrators
2. Which customers to assign to which concentrators
3. How to interconnect the terminals within a customer site

Each customer site would have a leased (or dial up) connection to the nearest concentrator or IMP. The IMPs would be connected by a highly redundant network of high speed lines (including satellite connections). The concentrators would be connected to the IMPs by medium speed lines, perhaps with or without redundancy. The customers would be connected to their concentrators by nonredundant low speed lines.

Many formidable theoretical problems are encountered during the topological optimization of any network. These problems include:

Network Choice: In general, there are

$$\frac{\left(\frac{N(N-1)}{2}\right)!}{M!\left(\frac{N(N-1)}{2}-M\right)!}$$

ways of arranging M links among N nodes. Considering all possible designs by computer is out of the question and no known computationally feasible method exists for finding an optimal computer communication layout for a system with a sufficiently large number of nodes.

Discrete elements: Components usually are available in discrete sizes. Thus, line speeds can be at 2000, 2400, 3600,. . . ,9600,. . ., 50,000, bits/s, etc. This means an integer optimization problem must be solved. Except in special cases for centralized tree design, no theoretical methods now exist for problems of practical size.

Nonlinearities: Component cost structures, time delay functions and reliability functions are all nonlinear. Typical cost functions are neither "concave" or "convex" and no analytical methods are available to obtain optimal solutions for networks containing such elements.

However, in spite of these difficulties, enormous progress has been made in the understanding of computer communication networks and in the development of effective analysis and design procedures.

In this report we focus on three very basic design parameters that we must consider: the selection of the channel capacities; the selection of the channel flows; and the topology itself. The performance criterion is the average message delay and the optimization is done under cost constraints. The optimum selection of the channel traffic consists of finding those theoretical average message flow rates for each line that will result in a minimum average message delay; we are not describing the routing procedure that will, in fact, achieve these channel traffic values. We can now define four optimization problems that differ only in the set of permissible design variables. In each of these problems it is assumed we are given the node locations, external traffic flow requirements and channel costs and also the flow we use is feasible (i.e., it satisfies the capacity, conservation and external traffic requirement constraints).

Capacity Assignment (CA) Problem

Given: Flow and the network topology

Minimize: Average message delay

With respect to: Capacities

Under constraint: Total cost

Flow Assignment (FA) Problem

Given: Capacities and network topology

Minimize: Average message delay

With respect to: Flows

Capacity and Flow Assignment (CFA) Problem

Given: Network topology

Minimize: Average message delay

With respect to: Capacities and flows

Under constraint: Total cost

Topology, Capacity and Flow Assignment (TCFA) Problem

Minimize: Average message delay

With respect to: Topological design, capacities and flows

Under constraint: Total cost

These four problems are solved in various degrees of completion; that completeness depends strongly upon the form of the cost functions. To get some feeling for the solution to these problems we start with the simplest possible case, solve it, then attempt to solve a progressively more difficult problem. First, consider the simplest case of continuous linear costs, then a manageable case of continuous concave costs and, finally, a difficult case of discrete cost functions in which the permissible channel sizes are drawn from a discrete set.

THE CAPACITY ASSIGNMENT PROBLEM

One of the more difficult design problems is the optimum selection of capacities from a finite set of options. Although there are many heuristic approaches to this problem, exa:t analytic results are scarce. It is possible to find reasonable (even optimal) assignments of discrete capacities for, say, 200-node networks, but since these assignments are the result of numerical algorithms, very little is known about the relation between such capacity assignments, message delay and cost. Thus, to obtain theoretical properties of optimal capacity assignments, we first ignore the constraint that capacities are available only in discrete sizes and assume they are available in continuous sizes.

With this assumption we solve the CA problem in which network topology and traffic flow are assumed known and fixed. First, consider the case of linear capacity costs. This problem can be solved by forming the Lagrangian and minimizing it with respect to the capacities. The exact solution is in the form of a square root channel capacity assignment. The concave case can be solved iteratively by linearizing the costs and solving a linearized problem at each iteration. For a logarithmic cost function the CA problem yields a proportional capacity assignment. A more realistic cost function is a power-law cost function. The optimal solution to this problem is obtained by solving the Lagrangian and iterating to the solution.[17] We do have methods of solving the capacity assignment problem when the capacities may be chosen continuously.

In practice, however, the selection of channel capacities must be made from a small finite set. In this case, it appears that an approach based upon dynamic programming is perhaps best. However, the continuous optimization procedures described here do provide a means for selecting among discrete capacities in a suggestive way. In general, the selection directly from the discrete set of channel capacities is a difficult dynamic programming problem. Another suboptimal technique for the solution of the CA problem is the Lagrangian decomposition (LD). The LD method is suboptimal in the sense that it determines only a subset of the set of optimal solutions corresponding to various values of the maximum total delay. However, in the case where the topology of the network is a tree, then a simple and computationally efficient algorithm is known that optimally selects channel capacities for an arbitrary discrete set of costs.

TRAFFIC FLOW ASSIGNMENT PROBLEM

Here the capacities are given and the flows must be determined to minimize the average delay. This is a flow optimization problem which belongs to network flow theory. The Max-Flow, Min-Cut theorem is the foundation of this theory. In terms of network flow theory, the selection of the flows can be viewed as a multicommodity flow problem with a non-linear objective function. The method described gives an exact solution to the FA optimization problem which is computationally efficient. It is the flow deviation (FD) method which locates the global minimum. This method is described in Chapter 5 of Reference 7.

One of the most popular heuristic algorithms used in solving the Flow Assignment problem is known as the minimum link algorithm. This algorithm is conceptually simple and computationally very efficient. Its major drawback is that it is rather insensitive to queueing delays and, therefore, possibly far from optimum in heavy traffic situations.[36]

There is a simpler suboptimal method that produces a fixed routing procedure and often yields good results using much less computation. The class of networks for which this fixed routing algorithm is effective is for the case of "large and balanced networks." This algorithm, known as the Fixed Routing Flow Algorithm, will converge in a finite number of steps since there is only a finite number of fixed routing flows to be considered.

CAPACITY AND FLOW ASSIGNMENT PROBLEM

In the two previous sections optimum solutions to the CA and the FA problems were given. Unfortunately, when these two problems are combined, globally optimal solutions are not easily obtainable. Instead procedures are given to find local minima for the average message delay.

To find these local minima we begin with a feasible starting flow, calculate the optimum capacity assignment under linearized costs, carry out the FD algorithm to find the optimum flows, repeat the CA problem for these new flows and continue to iterate between the CA solution and the FA solution until we find a (local) minimum. This CFA suboptimal algorithm will converge, since there is only a finite number of shortest route flows.

The CFA problem may also be posed in its dual form, namely

CFA PROBLEM (Dual Form)

Given: Network topology
Minimize: Total cost
With respect to: Capacities and flows
Under constraint: Average message delay

This problem can be solved using the CFA suboptimal algorithm with minor modifications. A description of this algorithm is found in Chapter 5 of Reference 7.

For the discrete cost problem one of the following heuristic approaches may be used.

Approach 1: Solve, iteratively, a routing problem with fixed capacities, followed by a discrete CA problem with fixed flows, until a local minimum is obtained.

Approach 2: Interpolate discrete costs with continuous concave costs. Solve the corresponding concave CFA problem. Adjust the continuous capacities to the smallest feasible discrete values. Reoptimize the flow assignments by solving a routing problem.

Four algorithms are next introduced, the first three following Approach 1 and the last algorithm following Approach 2.

1. Minimum link assignment
2. Bottom up algorithm

3. Top down algorithm

4. Discrete capacity algorithm

These algorithms are discussed in Reference 36.

TCFA PROBLEM

The topology design problem can also be characterized as follows:

Given:

Locations of the hosts and terminals

Traffic matrix

Cost matrix

Performance constraints:

Reliability

Delay/throughput

Variables:

Topology

Line capacities

Flow assignments

Goal:

Minimize cost

The (heuristic) solution espoused by Kleinrock[7] to solve this problem is known as the concave branch elimination method (CBE). It is an iterative form of the CFA (Capacity-Flow Assignment problem) solution and takes advantage of the fact that channels may be eliminated (and, therefore, the topology changed) as the CFA algorithm proceeds. The algorithm proceeds as follows:

CBE Suboptimal Algorithm

Step 1. Select an initial topology.

Step 2. For each channel in the topology make some approximation for the cost as a function of capacity and line cost (i.e., a power law approximation or a piecewise linear approximation). For each iteration in Step 3 below, use a linearized value for capacity about the value of flow for that channel.

Step 3. Carry out the CFA algorithm. If, at any iteration, a connectivity constraint is violated, then stop the optimization and proceed to Step 4; otherwise, let the CFA algorithm run to completion and then proceed to Step 4.

Step 4. From the CFA suboptimal solution, select discrete approximations for the continuous capacities obtained. For example, the continuous capacity may be "rounded" to the nearest allowable, discrete value so that the maximum delay constraint is still satisfied. This will tend to change the total investment or cost.

Step 5. Conduct a final flow optimization by an application of the FA algorithm.

Step 6. Repeat Steps 3 through 5 for a number of feasible random starting flows (by selecting random initial lengths with a shortest route flow).

Step 7. Repeat Steps 1 through 6 for a number of initial topologies.

The number of repeats in Steps 6 and 7 depends upon how many dollars one is willing to spend to find the solution. Experience with the ARPANET indicates that 20 to 30 repeats at Step 6 and a few (roughly five) initial topologies (of which one is fully connected and the others are highly connected) at Step 7 yield good results.

There are other approaches to solving this problem. One of these is known as the branch exchange method. Starting from an arbitrary feasible topology, a class of local transformations is defined in which one branch is removed and some new branch is added, so that feasibility (e.g., 2-connectivity) is preserved. Then a (simple-minded) CFA problem is solved using a minimum fit procedure. If this results in an improvement, then the transformation is kept, otherwise it is rejected. This procedure is carried out until the set of local transformations has been exhausted. An extension of the branch exchange method is the cut saturation method. This method reduces the set of local transformations to those that are good candidates for improving the throughput cost performance. It begins with a tree (or other low connected topology) and identifies the critical cut set resulting from an FA solution. It then adds a channel across this cut (or increases the capacity of a channel in this cut). This is repeated until a feasible topology is obtained. The procedure then continues, but, in addition, at each step the least utilized channel in the network is removed (if feasible). After a given number of iterations, the algorithm terminates. The key to all of these heuristic algorithms is the availability of efficient methods for evaluating topologies along with the random generation of several starting topologies (to search for many local minima).

Both the BXC and CBE methods have some shortcomings. For example, the BXC method requires an exhaustive exploration of all local topological exchanges and tends to be very time consuming when applied to networks with more than 20 or 30 nodes. The CBE method, on the other hand, can very efficiently eliminate uneconomical links, but does not provide for insertion of new links. To overcome such limitations, new methods derived from BXC or CBE have been proposed.

The cut saturation method can be considered as an extension of the BXC method. Rather than exhaustively performing all possible branch exchanges, it selects only those exchanges that are likely to improve throughput and cost. In particular, at each iteration a routing problem is solved, the saturated cut (i.e., the minimal set of most utilized links that, if removed, leaves the network disconnected) is found and a new link is added across the cut, and the least utilized link is removed. The selection of the links to be inserted or removed depends also on link cost.

The concave branch insertion method identifies and introduces links which provide cost savings under a concave cost structure. The method can be efficiently combined with the CBE method to compensate for the inability of the latter to introduce new links.

In some applications with very irregular distributions of node locations, or with constraints which are difficult to formulate analytically, network design can be greatly enhanced using person-computer interaction. To this end, interactive design programs have been developed in which the network designer can observe (and eventually correct) the topological transformations performed by the computer and displayed iteration after iteration on a graphic terminal.

In general, the selection of the appropriate algorithm will depend on the cost-capacity structure, presence of additional topological constraints, degree of human interaction allowed and, finally, tradeoff between cost and precision required by the particular application.

CENTRALIZED NETWORK DESIGN

Here we focus on the simplest type of network: a centralized system with all messages flowing inward to some central processing facility. The centralized network model described here applies to two very important problems:

- The terminal layout problem in which terminals are to be connected in multidrop fashion or multipoint to a specified concentrator
- The centralized network problem in which concentrators themselves are connected to a central processing facility

The two problems, terminal connection and concentrator connection, are really the same, although they may sometimes differ in the network hierarchy.

Simple structures for the design of a centralized network are often based on a shortest tree (often called a minimal spanning tree). Such a tree has the smallest total number of miles of communication lines of all possible trees. In general, the use of a tree in a centralized design may not provide the minimum cost solution. However, it can be shown that if all link costs are convex functions of capacity (this implies continuous functions), then some tree solution is optimal. On the other hand, the use of a shortest tree connecting all communicating points even under the simplest and most ideal conditions may still not be optimal. This is because one can add a set of S additional points, called Steiner points, to an N-node system and then find the shortest tree connecting the N+S points. If the new points are chosen appropriately, the resulting tree is called a Steiner minimal tree and the total length of all lines in the Steiner tree is less than the total length of all lines in the ordinary shortest tree without Steiner points. A simple example of a shortest tree and a Steiner tree is shown in Figure 19. Physically, a Steiner point corresponds to the location of a multiplexer, or concentrator, and although some procedures are available for finding good locations, this problem is presently tractable only by using heuristic procedures.

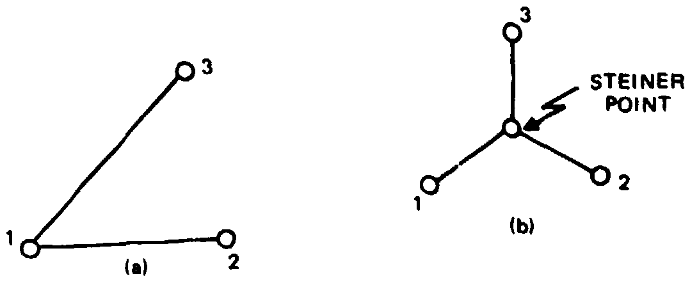


Figure 19. (a) Shortest tree (minimal spanning tree)
(b) Steiner minimal tree.

The most economical topology, even for a centralized network, may not be a tree. In practice, there is a finite set of choices for line capacities and, thus, line costs are not convex functions of capacity. In this case, a nontree topology will often be more economical than any tree topology.

In describing techniques developed for multipoint or centralized network design we first describe an algorithm due to Chandy and Russell using a branch-and-bound technique that provides the minimum cost solution to this constrained network design problem. We then focus on heuristic algorithms that generally provide suboptimum solutions quite close to the optimum with the desirable property of considerable savings in computation time.

Without constraints, the minimum cost network reduces precisely to the minimum spanning tree of the network of nodes to be connected together. The minimum spanning tree is, by definition, the spanning tree, a connected network containing all the nodes and no circuits or closed paths, the sum of whose link costs is a minimum. There are several algorithms to find the minimum spanning tree. One of these is based on the Kruskal algorithm. It is known as the Chandy-Russell algorithm. It is discussed in Chapter 9 of Reference 37.

Algorithms for finding the optimum constrained multipoint network may sometimes require very large computer running times. Now we consider some heuristic approaches. Three of these algorithms are the Esau-Williams algorithm, the Prim algorithm and the Kruskal algorithm. These are described in Chapter 9 of Referenc3 37. All three produce a minimum spanning tree solution when constraints are removed. In that case they then differ in their running time requirements. In the constrained multipoint design problem they produce somewhat different designs as well. Experience has shown that the Esau-Williams algorithm generally provides network designs closer to optimum.

The Esau-Williams algorithm essentially searches out the nodes that are furthest from the center (in a cost sense) and connects them to neighboring nodes that provide the greatest cost benefit. Prim's algorithm does the reverse: initially it selects the node closest to the center (again in a cost sense), then connects in those nodes that are closest to those already in the network. Kruskal's algorithm simply connects the least cost links, one at a time. For application to the multipoint problem, constraints have to be checked as a possible connection is made.

Other problems encountered in designing centralized computer networks are finding solutions to the following problems:

1. Number of concentrators
2. Terminal assignment - terminal to which concentrator
3. How to connect terminals
4. Concentrator location problem - ADD and DROP algorithms

5. How to connect concentrators to center

6. Clustering problem

Solutions to these problems are discussed in Chapter 9 of Reference 37.

DISTRIBUTED NETWORK DESIGN

A variety of algorithms have appeared in the literature that focus on least cost topological design of distributed networks. In most cases concentrator locations are assumed known. It is then necessary to choose the set of links, i.e., find the communications network that produces a least cost design subject to various constraints.

Here we focus on the cut saturation algorithm that assumes link capacity to be given and the same for all links in the network. It iteratively finds the least cost distributed network for a specified throughput, subject to time delay and reliability constraints. This algorithm is described in Chapter 10 of Reference 37. The cut saturation algorithm operates to relieve the most heavily congested portion of the network. It consists of five basic steps in any one iteration:

1. Routing. For a given network design, the optimal link flows are found that minimize overall average time delay.

2. Saturated cutset determination. Once the optimum flows are found, the links are ordered according to their utilization. The links are then removed, one at a time, in order of utilization. The minimal set that disconnects the network is called a saturated cutset.

3. Add-only step. This adds the least cost links to the network that will divert traffic from the saturated cutset.

4. Delete-only operation. This step in the algorithm eliminates links from a highly connected topology. One link at a time is removed at each iteration; the one removed is the most expensive and least used.

5. Perturbation step. Once a desired throughput range has been attained, the network links are rearranged, using Add-only and Delete-only operations to reduce the cost.

Another method is the branch exchange method. It iteratively adds, deletes or exchanges links and computes the corresponding cost and throughput variations. If the result of a topological modification is favorable, it is accepted. The procedure is exhaustive and terminates when no more improvement is possible. The branch exchange method is much more time consuming to run than the cut saturation method. (The cut saturation method is selective rather than exhaustive in its choice of links to be added or deleted.)

Other algorithms for solving this problem are given in the section on the TCFA problem.

The modeling, analysis and topological design areas for computer networks have all made significant progress in recent years. General network structures with several hundred nodes now can be handled, while specialized structures (such as centralized trees) with thousands of nodes are tractable. Effective design for small networks of less than about 25 nodes, is routine using the techniques that have been discussed previously. Design of networks with appreciably more nodes requires a high degree of both analytical and programming skill. Among the major open problems for large network design using present philosophies are the problems of clustering and partitioning. It appears that these problems will be well in hand in the near future, but that new design philosophies will have to be developed for general network structures with significantly more than one thousand nodes.

CHAPTER 4

PROTOCOL VALIDATION TECHNIQUES

INTRODUCTION

Most of the work on formal specification of protocols focuses on the logical correctness of the protocol and not on the service it provides. Approaches such as state diagrams, Petri nets, grammars and programming languages have been applied to this problem. These techniques may be classified into three main categories: transition models, programming languages and combinations of the first two.

PROTOCOL MODELS

Transition models are based on the observation that protocols consist largely of relatively simple processing in response to numerous "events" such as commands (from the user), message arrival (from the lower layer) and internal timeouts. Consequently, state machine models of one sort or another with such events forming their inputs are a natural model. Models falling into this category include state transition diagrams, grammars, Petri nets and their derivatives, L-systems, UCLA graphs and colloquies. However, for protocols of any complexity, the number of events and states required in a straightforward transition formulation becomes unworkably large.

Programming language models are motivated by the observation that protocols are simply one type of algorithm and high-level programming languages provide a clear and relatively concise means of describing algorithms. A major advantage of this approach is the ease in handling variables and parameters which may take on a large number of values.

Hybrid models attempt to combine the advantages of state models and programs. These typically employ a small-state model to capture only the main features of the protocol (e.g., connection establishment, resets, interrupts). This state model is then augmented with additional "context" variables and processing routines for each state. These context variables and processing routines are implemented in the programming language. In these hybrid models, the actions to be taken are determined by using parameters from the inputs and values of the context variables. The action at each major state is calculated using these inputs and values according to the processing routine specific for this state.

Finite State Machine Model

Finite state machines were proposed quite early to specify protocols. A single finite state machine can be used to describe the global state of the protocol or, alternatively, one machine can be used for each party. Each part of a protocol residing at a single process is called a party of the protocol, i.e., a party is a portion of a process which is relevant to a protocol. The single machine and the coupled machine models are theoretically equivalent and they are theoretically applicable to any protocol having finite parties and a bounded number of topologies. When a protocol with a multiple topology characteristic is represented by this model, a different set of global states corresponds to each permitted topology. Topology evolution can be represented by allowing transitions between states of different sets. Since the number of global states is finite, only a finite number of topologies can be represented. More detailed description of the use of finite state machines can be found in References 38 and 39.

Petri Nets And Related Models

Petri nets are graphs representing interacting processes contending for resources. Petri nets consist of transitions and places, interconnected by edges. Transitions model processing while places model the state of the system. The dynamics of Petri nets are described by tokens flowing from places to places across transitions. The presence of tokens on places causes transitions to fire, thereby causing some tokens to be placed in the output places of the transitions. The primary issue in Petri net research has been Liveness and Safeness. These two properties deal with deadlocks, traps and other control flow anomalies. Little attention has been paid to timing issues with exception of some work done on timed Petri nets.

The theoretical applicability of Petri nets as a descriptive model is broader than finite state machines. Some protocols having an infinite number of states can be represented by a Petri net in which the number of tokens can grow without limit. However, the Petri net model is not universal because certain party characteristics are not represented in this model. Petri nets are convenient for representing protocols which can operate with various amounts of some resources (e.g., number of buffers, etc.). Petri nets will also be convenient for representing parties in which several events may occur in arbitrary order.

The principal practical shortcoming of Petri nets (as well as state machines) is the rapid growth of the graph with the complexity of the protocol. To alleviate this, enhancements and generalizations of the basic model have been proposed and used to represent protocols. The enhancements result in a more compact notation and also have a broader theoretical applicability as a descriptive tool than the basic Petri net, but their generality makes them more difficult to analyze.

Another shortcoming of many protocol validation techniques, including finite state machines and Petri nets, is the fact that they do not keep track of time. Common Petri nets do not express time intervals assigned to the duration of various operations, e.g., time-out intervals. However, the definition of a Petri net can be extended. Petri nets with time intervals assigned to the transitions are called timed Petri nets. By extending the principles of Petri nets new formal means have been elaborated, namely the evaluation nets (E-nets). The extension consists in introducing additional types of nodes and attribute tokens and blocking transitions by logical conditions.

High Level Programming Languages

In this mode, each party is represented by a formal description similar to a high level program. Since these languages are universal, they permit the representation of any party characteristic. The characteristics of a party are given by the (possibly infinite) set of all possible pairs of incoming-outgoing message sequences, i.e., the characteristics describe all possible behaviors of the party. However, in the standard way in which these languages are used, only simple topologies can be represented. In practice, standard high level programming languages are convenient for representing numbers, data, variables, counters, etc., but not complex control structures. Therefore, this model was used mainly to represent the data transfer aspects of protocols while the graph models (state machines and Petri nets) were used mainly to represent the control aspects (synchronization, initialization, etc.) for which they are more convenient.

PROTOCOL VALIDATION

Protocol validation assumes there is a clear definition of protocol performance goals or the capabilities to be provided by the protocol to its users. Typical computer communication protocols perform both data transfer and control functions. For data transfer, performance goals include avoiding loss, duplication or damage of messages transmitted and delivering them in the proper sequence. For control functions, reliability goals involve the proper initialization and synchronization of control information on both sides of a connection. The possibility of deadlock and the consequences of protocol failures must be considered in assessing protocol reliability.

Transmission medium characteristics provide another important input to the protocol validation task. The operation of a communication protocol can be modeled by two automata connected by a transmission medium. The automata receive commands or events from their respective users and from each other via the transmission medium. Protocol validation consists of demonstrating the fulfillment of constraints on the operation of the composite system. These constraints may reflect the reliability performance goals, i.e., validation of the protocol in its potential operating environment.

The following points describe different aspects of protocol operation. Protocol validation can be considered the analysis of these different aspects and the comparison of the results obtained with the operational requirements.

Reachability Analysis

The basis for all subsequent validation aspects is an analysis of the possible transitions of the overall system. The reachability analysis yields the transition diagram of the overall system.

Deadlocks

A deadlock is characterized by a state or set of states of the overall system, reachable from the initial state of the system, for which no further transition is possible. Deadlocks must be avoided, since once the system has arrived in a deadlock state, it is blocked forever.

Liveness

A state is live if it can be reached from all states of the overall system that are reachable from the initial state. Usually a protocol contains a so-called "home state," or "steady state," which is live and from which all pertinent operations of the protocol can be reached.

Loops

Each protocol with a home state contains a loop in the transition diagram of the overall system starting at the home state and leading back to it. Usually, there are other loops necessary for the operation of the protocol. In addition, during the design phase of a new protocol, other loops might be found in the transition diagram. They are undesirable since their execution does not advance the useful processing of the protocol. Depending on the relative speed of different operations, these loops could be followed by the system an unlimited number of times.

Self-Synchronization and Stability

A system is self-synchronizing if, started up in any possible state of the overall system, it always returns, after some finite number of transitions, into the normal cycle of operation including the home state. This property is important for error recovery in an unreliable environment where, for example, the transmission medium does not always function properly or one station does not follow the prescribed protocol due to a software or hardware bug or the protocol is not properly initialized. The self-synchronizing property implies protocol stability, because it ensures that the protocol reverts directly to its normal mode of operation after any initial or intermittent perturbation in the synchronization of the two communicating subsystems.

VALIDATION METHODS

A number of formalisms have been used to specify protocols, including flow charts, programming languages, state diagrams, state transition matrices, Petri nets, UCLA graphs and prose. A table taken from Reference 40 comparing protocol validation techniques is given in Table 7.

There are two main categories of protocol validation: reachability analysis and program proofs. Reachability analysis is based on exhaustively exploring all the possible interactions of two (or more) entities within a layer. It is particularly straightforward to apply to transition models which have explicit states and/or state variables defined. For a given initial state and set of assumptions, this type of analysis determines all of the possible outcomes that the protocol may achieve. The major difficulty with this technique is "state space" explosion.

The program proving approach involves the usual formulation of assertions which reflect the desired correctness properties. The basic task is then to show that the protocol programs for each entity satisfy the high level assertions. These assertion methods are more successful with data transfer aspects of protocols since they can represent whole classes of sequence number in single symbolic assertions. A major strength of this approach is its ability to deal with the full range of protocol properties to be verified, rather than only general properties.

Principal authors	Modeling formalism	Analysis technique	Focus	Protocols analyzed	Conditions checked	Hard Steps	Additional work needed
Danthine Bremer	Partitioned FSA (plus algorithms)	Compatible paths	Control	Cyclades TS EDF-GDF	Deadlock	Define FSA	Data transfer looping asymmetry
Brand Joyner	Algorithms	Symbolic execution assertions	Data trans-fer	micro I/Q ARQ	Deadlock looping data tıansfer	Define assertions	Initialize complex media user commands
Rudin West Zafiropulo	FSA	Compatible paths composite FSA	Control	X.21	Deadlock completeness	Define FSA, assertions	Complex media looping data transfer
Bochmann	FSA with variables, algorithms	Composite FSA assertions adjoint states	Both	X.25 ARQ HDLC	Deadlock looping liveness	Define FSA, Prove assertions	Automation, Complex media
Harangozo	Formal grammars	—	Both	HDLC	—	Define grammar	Flow control user commands verification
Hajek	Algorithms	Composite FSA	Both	ARQ ARPA TCP	Deadlock looping termination	Define algorithm, assertions	Complex media, state explosion
Symons	FSA augmented Petri nets	Composite Petri net simulation	Both	ARQ ARPA NCP	User provided	Define FSA, assertions	Automate assertions
Gouda	FSA	Composite FSA compatible paths	Control	ARQ	Deadlock boundedness	Define FSA, Form proof	Complex media, protocols automation

Table 7. Comparison of some protocol verification techniques.

A third approach is a hybrid approach. This could combine the advantages of both of the above techniques. By using a state model for the major states of the protocol, the state space is kept small and the general properties can be checked by an automated analysis.

Global State Generation

One of the most common validation techniques is exhaustive global state generation. The theoretical applicability of this technique is limited to protocols with bounded number of topologies and finite state parties. The practical applicability is limited to very simple topologies (up to six parties). An advantage of this technique is that the state generation can be easily mechanized and several properties can be automatically tested. Sometimes, properties of the total space can be validated by generating a small subset of the states. This can greatly increase the applicability of the technique.

Assertion Proving

Another common protocol validation technique is assertion proving, which is applied to the protocol description as if the description were a parallel program. This technique is usually applied to protocols modeled in high level programming languages but theoretically can be applied to any other model. The usual way of applying this technique is to attach a predicate of the variables' values to certain points in a program and prove that whenever the program reaches these points the predicate is true. The method can be generalized to protocols with an unbounded number of topologies provided that the desired predicate and the (possibly unbounded) sets of points can be expressed by bounded expressions. In practice, assertion proofs are applied mainly to simple topologies, but sometimes have complex parties. Since the construction of proofs may require creativity, this technique cannot be fully automated. However, quite powerful theorem provers have been constructed which are capable of automatically proving many of the required properties. While global state generation is more convenient in proving control properties (e.g., that certain events will or will not occur), assertion proving is used mainly in proving data transfer properties, particularly in protocols involving parties with large or infinite state space. The two techniques can be also combined to capitalize on the advantages of each.

Induction Over the Topology

By this technique, the holding of a property or the occurrence of an event is proven by showing certain conditions will propagate throughout the topology. The use of induction over the topology is theoretically applicable to protocols of any characteristics, provided the topology is unbounded and can be represented as described in Reference 41.

Adherence To Sufficient Conditions

In this technique, the protocol is designed so each design step is done satisfying conditions sufficient to guarantee the required properties. That is. instead of designing a protocol and later proving its correctness, this technique is aimed at directly designing one which is correct by construction. This technique can be used in any topology and party characteristic and its main advantage is that it is easy to apply and that correctness is directly guaranteed. Its main shortcoming is that sufficient conditions could be too strong, i.e., there may be many correct protocols that will be rejected because they do not satisfy the sufficient conditions. On the other hand, tight sufficient conditions (or preferably necessary and sufficient conditions) are usually complex and difficult to find.

Partial Specification and Validation

Depending on the specification method used, only certain aspects of the protocol are described. This is often the case for transition diagram specifications which usually capture only the rules concerning transitions between major states, ignoring details of parameter values and other state variables.

SARA and GMB

The need for formal methods of verifying communication protocols has motivated a great deal of research in recent years. The SARA Graph Model of

Behavior (GMB) provides a powerful tool to model and validate protocols. Generally, the GMB control flow primitives are sufficient to clearly express protocol interactions at an abstract level. However, when necessary, the GMB's data flow and interpretation models are available to remove ambiguity. Modeling in the control domain often permits completely automatic analysis of a protocol, but runs the risk of an explosion in the number of control states. Use of the data and interpretation domains makes available abstractions which significantly reduce the number of control states while providing enough information to permit semiautomatic validation.

There is a duality between state-transition diagrams and GMBs. In a GMB, states are modeled using control arcs. The fact that a system is in a particular state is modeled by the presence of a token on the arc corresponding to that state. Most protocols contain a small number of distinct states, making it feasible to model all states entirely in the control domain.

Signals flowing can also be modeled using control arcs. A separate control arc may be used to model each separate signal. For example, in the X.21 interface, placing a token on a control arc used to model a signal represents a circuit changing its state to satisfy the meaning of that signal.

Transitions are modeled by control nodes which can absorb tokens from one state and place tokens on its successor state. Receive transitions absorb tokens from the arcs modeling the incoming signals. The act of absorbing a token from an arc models the fact that a component has reacted to a change on its input circuits. Send transitions place tokens on the arcs modeling the outgoing signals. The act of placing a token on an arc models a component changing the status of an output circuit in a manner which satisfies the meaning of an outgoing signal.

This information is discussed in Reference 29.

CHAPTER 5
AVAILABLE DESIGN TOOLS

This chapter presents four currently available design tools. A synopsis of their capabilities is given in Table 8. Two of these tools, MIND and GRINDER, are software packages to design, analyze and maintain telecommunications networks. The other two, SARA and PCD, are for designing hardware and software systems.

MIND

Introduction

MIND, the Modular Interactive Network Design System, is an interactive system that allows the user to design, analyze and maintain centralized telecommunications networks. (See Figure 20a.) Since point-to-point networks or subnetworks are special cases of multipoint networks, MIND is also useful in the design, analysis and maintenance of such architectures. (See Figure 20b and Figure 20c.)

MIND Architecture

A high level structure diagram of MIND is given in Figure 21 illustrating its major functional blocks and their interrelation. All communication with the user is done via an interactive executive program which maintains a dialogue with the user and controls the execution of the background applications. The applications are divided primarily into two groups. The first group includes Network Editor and Design modules and allows the user to generate and maintain centralized ıetworks. The second group contains the Multipoint Line Simulator which allows the user to study the response time/throughput performance of centralized networks by modeling traffic loading and details of line protocol.

Design Tool	Tool Elements	Capabilities	Models
MIND (Modular Interactive Network Design) System	Network Editor	Edit Network Data Base: 1. Define network (nodes, links) 2. Save and retrieve 3. Modify 4. Display	Nodes connected by links. Centralized net: terminals connected by multipoint circuits to concentrators. Concentrators connect to central node (one).
	TOPO	Topological Design and Analysis: 1. Designs and evaluates multipoint line layouts. 2. Gives cost, loading, reliability	AT&T Tariff Schedule. Serial reliability model with parallel devices for redundancy.
Network Analysis Corp.	MLSS	Multipoint Line Simulation System: 1. Allows user to enter model of the network and traffic load and obtain waiting times, response times.	Traffic: Length, intensity Protocols: ACK, control, polling Net: Line speed, processing times, propagation delay
		2. Protocol comparison	Line: Half duplex Errors: Line errors, terminal errors (random) Polling: With/without ACKS discrete, string
POD (Performance oriented design)		Models the performance of computer based systems at various stages in the system life cycle.	Hardware and software modeled by physical characteristics (speed, storage, number of CPUs), interconnections. Workload model = number of jobs (priority and arrival rate).

Table 8. Design tool capabilities.

Design Tool	Tool Elements	Capabilities	Models
BGS Systems		Evaluates throughputs, response times, bandwidths, utilization and component loading. Uses analytic queueing theory	

	Front End		Hardware = "Black Boxes"
			Software = visible internal structure

	Life Cycle Modeling	Enable models to evolve and retain continuity with earlier versions. Interactive modeling environment	
SARA (System Architects' Apprentice) UCLA ARPANET		Requirement driven design of modular, concurrent, hardware and software systems.	Explicit system and environmental models required. Multilevel partitioning - each subsystem treated recursively as a new synthesis problem. Models use: Modules - totally contained structures. Socket - provides communication to module. Interconnection - bridges two or more sockets.
GMB (Graph Model of Behavior)		Models communication protocols in three modeling domains: control, data, interpretation.	Control Graph: Nodes = events; Arcs = precedence relationships among events. (Static)

Table 8. Design tool capabilities. (Continued)

Design Tool	Tool Elements	Capabilities	Models
UCLA			Data Graph: datasets = static collections of data.
ARPANET			Processors = data transformers Data Arcs = data paths Data Formats = PL/1 declarations Data Transformations = PL/1 algorithms.
GRINDER (Graphical Interactive Network Designer)		Interactive software package for communications network designers.	
	Network Editor	Maintains a network data base which can be modified, viewed, saved.	Data Base = nodes, arcs, links
	Traffic Editor	Reads and stores traffic files.	Backbone traffic described by by an NxN matrix.
ARPANET		Produces random traffic patterns.	
	Tariff Editor	Provides tariffs for all line costing routines. User may add own tariff structures.	Piecewise linear tariff. (cost per mile) TELPAK (speed dependent)
	ACUT	Automated cut saturation algorithm. Designs packet switched network to meet a set of user specified performance measures (throughput, delay, connectivity).	User specifies initial network configuration, traffic matrix.
	GLAD	Designs local access networks. Program seeks to connect the terminals to the centers as efficiently as possible, placing concentrators at eligible sites where cost is advantageous. Can also handle two level hierarchy (no concentrations).	Assumes network hierarchy: terminals, concentrators, centers (backbone nodes).

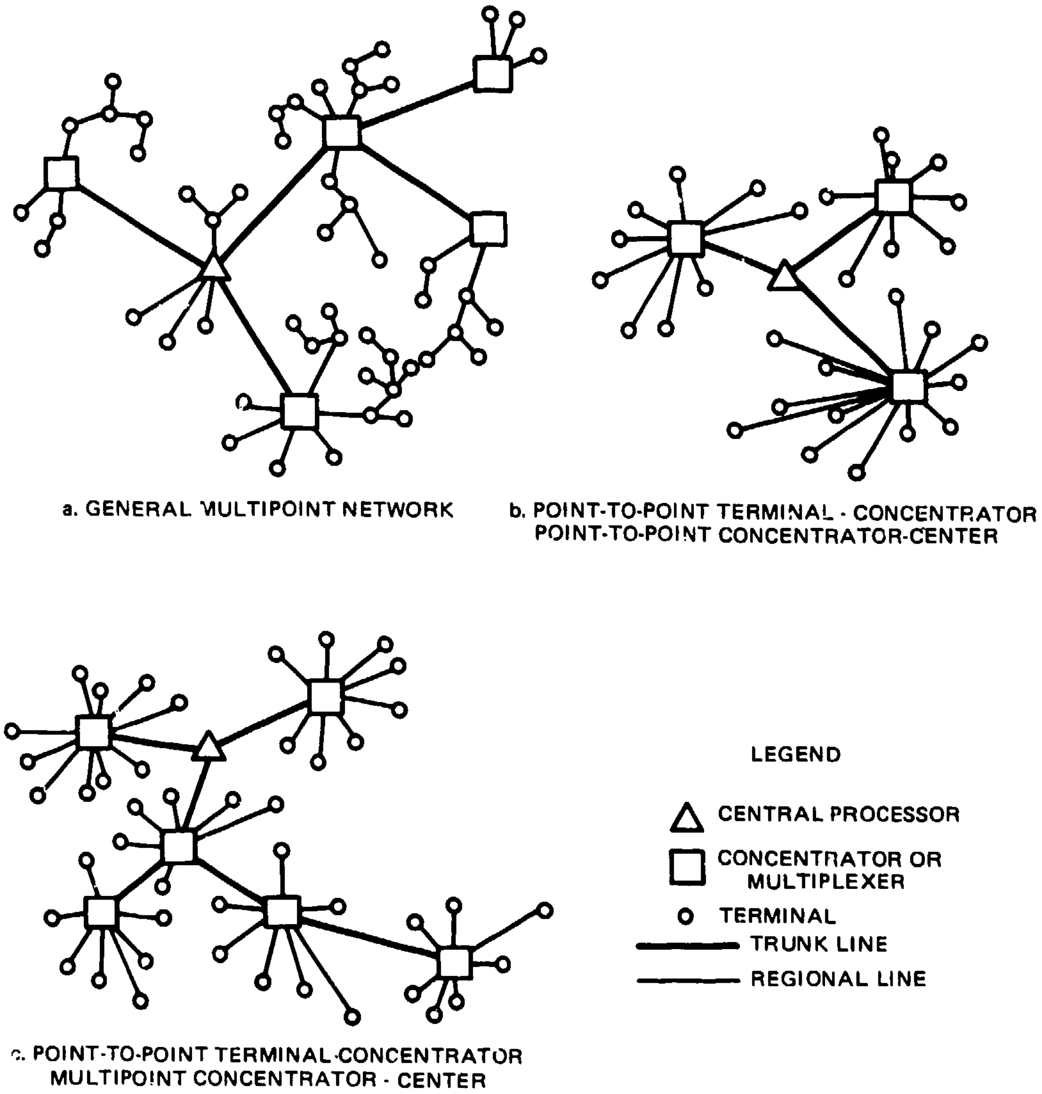
Table 8. Design tool capabilities. (Continued)

Design Tool	Tool Elements	Capabilities	Models
		A local net is divided into one or more partitions. Each consists of a center node and associated terminal nodes. Each partition is designed independently.	Presently handles only tree structures. (Will add loops and dual homing in the figure.) Handles up to 500 nodes. Input Data: Nodes and traffic Design constraints Tariff selection
	HIER	Allows user to evaluate 2 level hierarchical designs: backbone and local access nets. Makes use of ROUT and GLAD routines:	Input Data: Network configuration = Host nodes Terminal nodes Backbone nodes Interconnect nodes
	DSAG	Deletes low level links to backbone nodes.	
	ASG/DUHM	Assigns low level nodes to nearest backbone node. Assigns capacity to links.	
	CBC	Selects backbone nodes using a clustering technique.	
	CBA	Selects backbone nodes using an adding techniques.	
	BNA	Statistical Analysis Module. Gives summaries of cost, traffic, devices and hardware associations.	
	LAL	Statistical Analysis Module. Calculates total cost of local access lines.	

Table 8. Design tool capabilities. (Continued)

Design Tool	Tool Elements	Capabilities	Models
	LHL	Calculates total cost of local access hardware.	
	CST	Calculates cost of backbone switches.	
	ROUT	Determines the maximum throughput and optimal routes in a packet switched network for a given source-to-destination traffic pattern, under, end-to-end delay constraints. Uses the extremal flow algorithm.	Packet switched network Inputs: Source-Destination traffic pattern
	END	Postprocessor that evaluates and generates end-to-end delay stats.	Allows three traffic classes. User specifies traffic level, packet size, priority structure.
	HOP	Postprocessor that evaluates min-hop distance statistics.	Strictly topological (ignores traffic flows, capacity, line lengths).
	CAND	Evaluates new links for reducing maximum and average min-hop distance.	
	CUT	Postprocessor aids in identifying network bottlenecks.	

Table 8. Design tool capabilities. (Continued)



a. GENERAL MULTIPOINT NETWORK

b. POINT-TO-POINT TERMINAL - CONCENTRATOR
POINT-TO-POINT CONCENTRATOR-CENTER

c. POINT-TO-POINT TERMINAL-CONCENTRATOR
MULTIPOINT CONCENTRATOR - CENTER

Figure 20. Network architectures.

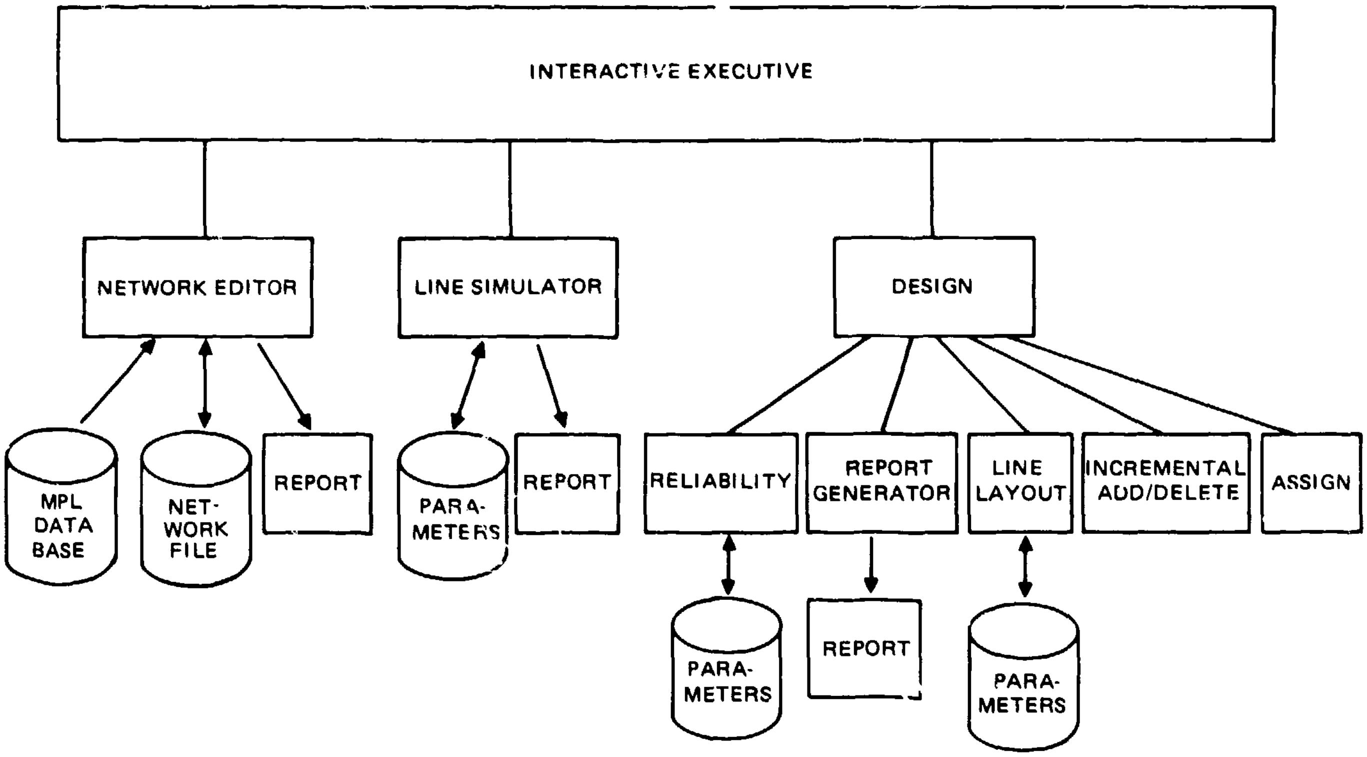


Figure 21. MIND architectures.

Network Editor (EDIT)

The Network Editor provides the user with capabilities to edit the network data base. The editor provides commands to:

1. Initially define a network data base by adding data elements (nodes and possibly links) and assigning values to the properties associated with each node and link.

2. Save copies of the network data base on disk and retrieve these copies at a later time.

3. Modify the network data base by adding and/or deleting nodes and/or links by assigning different values to the properties of the nodes or links of an existing data base.

4. Selectively display the property values of individual nodes or links on the user's terminal.

The network is modeled with two data elements: nodes and links. A node is either a source or destination of traffic (such as the center node or terminal devices) or it moves information through the network (such as a concentrator). Links are communication lines in the network between pairs of nodes. Each node and link has associated with it a set of properties which define what each node and link does in the network.

MIND will generate a design of communication lines connecting nodes in a centralized network. All communication in the network is considered to take place through the center node which is the highest level in a centralized network. There should be only one center node in the data base. The center node may be an actual central site or a backbone node in a larger network. The center node may also be more than one computer and the associated communication and peripheral equipment.

The second level has one or more devices which are called concentrators. At least one such device must be present in the model since all multipoint circuits must terminate at a concentrator, not at the center. In the situation where lines do actually terminate at the center, the user must enter a separate node with the same location as the center to act as a concentrator.

The lowest level of the network is the connection of terminals. Terminals, which serve as sources and destinations of traffic, are interconnected by multipoint circuits. A terminal node can represent one or more terminal devices at the same location, in the same area code/exchange or even in the same city. The user must assign values to node properties to account for all the devices which are represented by the node.

Each node has the following properties associated with it:

1. Label
2. Street (STR): street address descriptor
3. City (CITY): city descriptor
4. State (STAT): 2 character code for the state (Post Office)

5. Area-Code/Exchange (ACEX): the 6 digit telephone company area code and exchange

6. Role (ROLE): the role the node plays in the network; currently a node can be a CeNTeR (CNTR), a CONCentrator or similar device (CONC) or a TERMinal (TERM).

7. Homing Point (HOME): for a terminal node, the homing point is the concentrator to which it is assigned; for a concentrator, the homing point specifies which trunks are to exit from the node to other concentrators or the center; the homing point of the center node is not significant and is ignored by MIND.

8. Location Identifier (LOC): the user can specify all the nodes which are considered to be in the same location; the design program will not put more than one node with the same nonblank location identifier on the same circuit.

9. Receive traffic (RC): a measure of the amount of information received by the node. Units are kilobits per second.

10. Transmit traffic (TX): like RC, but the traffic transmitted by the node. Units: kbps.

11. Number of devices (NDEV): the number of terminal devices at that node.

12. Availability (AVAIL): the fraction of time the node is available; a measure of the node reliability.

13. Termination Cost (TER): the cost of termination equipment at the node site.

It is important that the user of MIND understand the concept of a link and a circuit in the network model. A circuit is a description of a multipoint line which connects terminals to a concentrator. The common carrier determines the cost the customer pays for such a circuit by the specification of the tariff. The carrier need not actually implement the circuit as it is specified. The cost is usually the minimum cost for the user and the carrier has the option to implement it in any fashion. MIND will represent the circuits in the design by adding links between the nodes on the circuit. The actual layout of the multipoint line will be the same as the minimum cost circuit as determined by the tariff module.

The link data element currently performs two functions: one, as input to the design program it helps define the network; and two, as the output from the design program it defines the design generated by MIND. As output from the design, a link can represent a span in a multipoint line or a trunk line (i.e., a connection between concentrators or between a concentrator and the center). As a span in a multipoint line, the link is a mechanism for assigning a cost to the circuit. Thus, if the user deletes a link from the design, the cost of the network will not necessarily be decreased by the cost of the deleted link. In fact, the cost of the network may increase as a result of dropping a node from the design.

Each link has a circuit identifier associated with it. A link can belong to (or be a part of) only one circuit. The identifier can be assigned by the user or by the design program and each circuit must have a unique identifier.

As input to the design program, the links serve to define a circuit or force a component of nodes to be placed on the same circuit. Each link has the following properties associated with it:

1. Endpoints: the two nodes at either end of the links
2. Circuit Identifier (CIRC): descriptor which specifies what circuit this link is on or, of it is blank, that the link is not yet a part of the circuit
3. Availability Fraction (AVAIL): the fraction of time the link is available; a measure of the link reliability
4. Length (LEN): the mileage between the two nodes which are at the endpoints of the link
5. Cost (COST): the cost incurred in the design by this link; the user should be aware that removing the link (i.e., deleting a node) may cause the design to be more expensive.

Multipoint Line Simulation System (MLSS)

The Multipoint Line Simulation System (MLSS) allows the user to enter a model of his/her network and traffic load and obtain an analysis of waiting times and response times in various parts of the network. The model may be finely tailored to the user's network by entering parameter values specific to the system at hand. The user is able to specify the traffic (length and intensity) distribution, protocol (acknowledgement scheme, handshaking, control messages) and network characteristics (e.g., line speed, number of terminals, processing times, propagation delays). The user also can obtain the values of key measures of response time performance such as waiting times for input and output messages and terminal response times. Empirical distributions as well as statistical averages are available.

The focus of MLSS is on modoling the multipoint line. Polling (determining whether or not a terminal is ready to send input), selecting (determining whether or not a terminal is ready to receive output), positive and negative acknowledgement, error occurrence and recovery and hardware delays in all devices are all modeled directly.

Simple models of the operation of the communication processor and central processor are also provided so that issues of response time can be examined in the context of a complete network.

MLSS was included in MIND primarily for the purpose of adding to the user's understanding of issues relating line loading to response time. With it the user can analyze issues such as the relationship between the number of terminals or the amount of traffic on a multipoint line and the response times users of that line can expect to receive. With this information in hand, the user can then set the appropriate parameters in the MIND line layout procedure and obtain the design of a network satisfying a given response time constraint. Another approach would be to obtain line layouts for a variety of loads and then use MLSS to examine their response time performance.

MLSS is primarily intended as a tool to provide response time statistics for a typical terminal rather than for a particular terminal on a specific line. Nonetheless, the user has the option of entering sufficient information to closely model the operation of a specific line for which he/she has detailed traffic information.

Other uses of MLSS include the comparison of protocols with different features and options, examination of the effects of altering the mix of traffic applications sharing the line and analysis of the sensitivity of response time to variations in traffic mix and network configuration. Detailed and exact analysis of a given situation is obtainable. Alternatively, an efficient approximate analysis can also be obtained of a large number of cases where only approximate information is available.

System Model

The system model is one of a transaction oriented system (e.g., inquiry/ response) where users contend for terminals and terminals share a multipoint line via polling and selecting. Delays suffered at higher levels of the network are modeled simply by a random distribution. The user is given a considerable amount of flexibility in parametrically defining this random distribution. The line is effectively half-duplex (or full-duplex operating in a half-duplex mode). Output has priority over input. MLSS also models the presence of broadcast messages. Broadcast messages have priority over inputs but not over outputs.

Figure 22 illustrates the life cycle of the modeled transaction along with the performance measures monitored by MLSS. Nine points in time are identified relative to the life of each transaction.

Time 1: The time the transaction arrives at the terminal. This may be a customer arriving at a service desk or a broker following up on a request for a quote, for example. Arrivals at a terminal are assumed to be independent of one another (Poisson).

Time 2: The time preservice begins. Preservice refers to all processing which occupies the terminal prior to the time the input message is ready to be transmitted to the center. This could include keying time and time for any preparation done at the terminal prior to depressing the "SEND" key. The time between Time 1 and Time 2 is spent waiting for the terminal to become available.

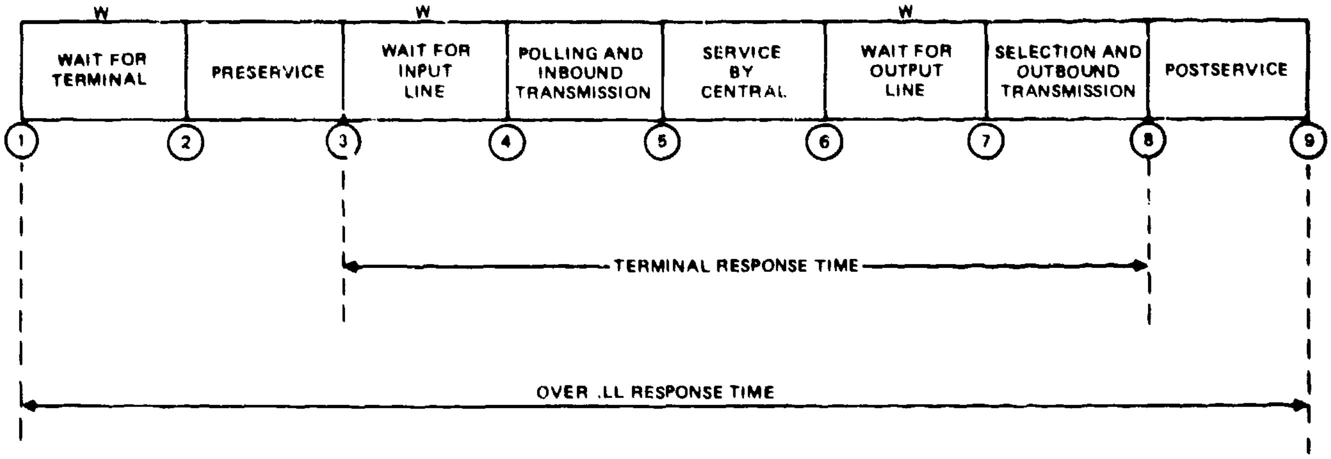


Figure 22. Message life cycle and performance measures.

Time 3: The time preservice ends. At this time, the transaction is in the system and ready to be transmitted. It is waiting for the terminal to be polled; i.e., it is waiting for the multipoint line.

Time 4: The time polling of the terminal and transmitting of the input message begins. The time between Time 3 and Time 4 is spent waiting for the multipoint line.

Time 5: The time transmission of the input message ends and service by the center begins. The time between Time 4 and Time 5 includes time spent polling the terminal (positive), response to poll, transmission of the input message and acknowledgement of the input message by the center. It also includes the time for timeouts and retransmissions due to the occurrence of errors in these processes. The multipoint line is occupied during this time interval. Contention for the center is modeled indirectly in the distribution of service times by the center.

Time 6: The time service by the center ends. At this point, the output message is ready to be sent to the terminal.

Time 7: The time selection and output transmission begins. The time between Time 6 and Time 7 is spent waiting for the multipoint line.

Time 8: The time transmission of the output message ends. The time between Time 7 and Time 8 includes selection of the terminal by the center, transmission of the output message and the time for timeouts and retransmissions due to errors occurring during these processes. This line is occupied throughout this interval. Time 8 marks the last point at which the transaction occup'es the line. The interval of time between Time 3 and Time 8 is the time required to go through the network. It is directly affected by line loading and is often an important constraint on the design of the network, in particular, on the layout of the multipoint lines. We refer to this time interval as the terminal response time.

Time 9: The time postservice ends. Postservice refers to all processing which occupies the terminal after transmission of the output message begins. This includes printing the output message (on hard-copy terminals) and any other wrapping-up done at the terminal before it becomes available for another transaction. Time 9 marks the end of the transaction. We refer to the interval between Time 1 and Time 9 as the overall response time.

Error Models

Two types of errors are modeled, line errors and terminal errors. The user may specify a bit error rate. Line errors occur randomly with a probability equal to the product of the bit error rate and the length of the message. If a line error occurs, the message is retransmitted. The user is also allowed to specify the maximum allowable number of retransmissions.

The user can also specify the probability with which terminal errors occur. Terminal errors are modeled as persisting for a sufficient length of time to cause the maximum number of retransmissions and subsequent abortion of the message.

Timeouts can also be set by the user for polling, selecting input message transmitting and output message transmitting. A timeout is also assumed to occur in the absence of an acknowledgement. In particular, a timeout occurs when a message is correctly received and no positive ackowledgement is used, a terminal error occurs or when polling or selecting without acknowledgements. The description of the MLSS protocol models in the following section elaborates further on this. The user specifies whether or not positive and negative acknowledgements are used in the system being modeled.

Protocol Models - Four types of polling are modeled:

- Discrete polling (unacknowledged). The center queries each terminal in turn if it has an input message which is ready to be sent. The terminal either responds negatively or with a positive response followed immediately by the input message.
- Acknowledged discrete polling. This is identical to unacknowledged discrete polling except when the terminal has a message ready to be sent, it gives a positive response to the poll without immediately transmitting the message. The center then sends a "SEND" sequence to the terminal and the terminal proceeds to transmit its input message.
- String polling (uninterruptable). The center sends a string of terminal addresses requesting input messages from the terminals. If a terminal has something to send, it informs the center before the latter completes sending the next terminal's address. After an input message is transmitted, the string poll picks up from where it left off. An output arriving during a string poll must wait for the polling sequence to be completed before it can be transmitted.
- Interruptable string polling. This is identical to uninterruptable string polling except arriving outputs can interrupt the polling sequence.

Two types of selection are modeled:

- Acknowledged selection. The center inquires if a terminal is ready to receive output. The terminal responds positively. The center then transmits the output.
- Fast selection. The center transmits a selection sequence, informing the terminal it is about to send a message and then immediately sends the output message to the terminal.

Random Distributions

The user is able to parametrically describe three types of random distributions as part of the overall model of the network and traffic. In particular, a constant, or piecewise-linear, distribution can be chosen to model message lengths (input, output and broadcast) and service times (preservice, postservice and center).

Available Parameters

Category COMM - Communication Processing Times. The parameters in this category allow the user to account for processing and hardware delays associated with the center and terminal during normal transmissions. Examples of times which should be included in the values assigned to these parameters are the time to execute software in concentrators polling the terminals and the time required to transfer a screenful of information from a slave terminal to the memory of its associated terminal controller. These include terminal processing time for a poll, terminal processing time for a selection, terminal processing time for input messages, terminal processing time for output, central processing time for poll, central processing time for selection, central processing time for input messages and central processing time for output messages.

Category CMES - Control Messages. The parameters in this category allow the user to specify the length of control messages used to implement the line protocol being modeled. These include the following length of

1. poll sequence
2. positive response to a poll request
3. negative response to a poll request
4. poll sequence header
5. poll sequence trailer
6. "SEND" sequence in acknowledged discrete polling
7. selection sequence
8. positive response to a selection sequence
9. positive acknowledgement of an input message
10. positive acknowledgement of an output message
11. negative acknowledgement of an input message
12. negative acknowledgement of an output message

Category SERV - Service Times. The parameters in this category allow the user to model preservice, postservice and time spent being processed in the center. The variables in this category form three groups of three variables, each of which defines a random distribution. They describe the distributions for the central service, preservice and postservice times.

Category ERR - Error Handling. The parameters in this category allow the user to model the occurrence of line and terminal errors in the simulated system. These include terminal error rate, bit error rate, poll timeout, selection timeout, input message timeout, output message timeout and maximum number of retransmissions before aborting message.

Category NET - Network Configuration. The parameters in this category allow the user to specify details of the network configuration. These include:

1. Number of terminals on the multipoint line
2. Terminal control unit to which the terminal is attached. This is significant in the situation where several terminals are controlled by a common TCU and the system does general polling. The user assigns an integer to each terminal. Terminals having the same TCU number will not be polled individually. The default is to keep each terminal separate.
3. Number of bits per character
4. Line speed
5. Average propagation delay
6. Modem turnaround delay at the central site
7. Input buffer length
8. Output buffer length

Category RUN - Simulation Run Options. The parameters in this category allow the user to control MLSS running time and output. These include maximum time allowed on this run, maximum number of simulated transactions, number of trace output lines, terminal queue switch and empirical distribution switch.

Category PROT - Protocol Options. The parameters in this category allow the user to choose the type of polling and selecting used as well as the type of acknowledgements used. These include type of polling, type of selecting, type of acknowledging used for input messages and type of acknowledging used for output messages.

Category TRAF - Traffic Specification. The parameters in this category allow the user to specify the length and frequency of input, output and broadcast messages. Message arrivals are assumed to be Poisson; i.e., messages are assumed to arrive independently of one another.

Topological Design And Analysis Module (TOPO)

This design module allows the user to generate and/or evaluate designs of a given network. A near optimal multipoint line layout can be obtained under the existing AT&T Multischedule Private Line (MPL) tariff subject to line loading constraints. Terminals can be assigned to the nearest home (concentrator, multiplexer or center). The effect of adding or deleting a terminal from the network can be evaluated and the analysis can be obtained. At any point the user can obtain a report on the cost and layout of selected circuits.

The Topological Design and Analysis subsystem, TOPO, allows the user to perform many useful functions directly related to obtaining a good line layout; i.e., a low cost interconnection of network devices satisfying a given set of constraints. It also contains post-processors which will analyze a given layout and evaluate commonly used figures of merit in the areas of cost, loading and reliability. Finally, several of its modules are specifically designed to deal with the problems which arise in maintaining a network; i.e., problems involving the modification of an existing layout rather than producing an initial layout.

Figure 20 shows the type of multilevel centralized networks treated by TOPO. The highest level of the network is called the center. All communication is assumed to take place to and from this location; communication among other locations is assumed to take place through it. The center may be an actual central site or a backbone node in a larger network; whichever it is, it is not significant to TOPO.

The second level of the network is comprised of one or more devices, which we will generically refer to as concentrators. At least one such device must be present in the model as all multipoint circuits must terminate at a concentrator, not the center. Situations where circuits terminate directly at the center are modeled by placing a concentrator at a separate node with the same location as the center.

At the lowest level of the network are the terminals which serve as sources and destinations of traffic. They are interconnected in multipoint circuits tying them to a concentrator.

A network, as modeled in TOPO, contains precisely one center, one or more concentrators and one or more terminals. Each terminal must be assigned to a concentrator.

TOPO generates multipoint circuits. Trunks are links between pairs of high level nodes. The user can enter trunks directly or, alternatively, they can be generated by the line layout procedure.

Assign Homes

The first step in the line layout process is to assign each terminal to a concentrator. The information is recorded in the HOME property of each node. This may be done in one of three ways, depending on the user's objectives. The user may prefer to make the assignment manually using the EDIT routines. In this way, the user can make assignments subject to loading constraints on the concentrators. In particular, he/she can control the total traffic, number of multipoint circuits and number of terminals assigned to each concentrator.

The second way of obtaining an assignment of terminals to concentrators is to use the REDO * command in the TOPO/LL module. This command will produce a complete line layout, assigning all terminals to multipoint circuits without regard to any previously existing assignments. As part of this process, each terminal will be assigned to a concentrator. The assignment is made strictly on the basis of minimizing cost. Of the three approaches, this is the one most likely to yield the lowest cost layout.

The third way of assigning terminals to concentrators is via the ASGH command in the TOPO base. This command will assign one terminal, a list of terminals or all terminals to concentrators on a "nearest neighbor" basis; i.e., each terminal will be assigned to the concentrator nearest to it. The command will also assign concentrators to the center.

Line Layout (LL)

The Line Layout subsystem allows the user to create and incrementally design multipoint circuits. The user can edit and use parameters to constrain the line layout to conform to limits on the number of devices and traffic on a circuit. He/she can also force associations between nodes on a circuit and to given concentrators. Thus, the user can take existing circuits into account and incrementally add to or modify an existing design. Alternatively, he/she can examine what can be gained by a complete reoptimization.

The Line Layout procedure can work from scratch or on some (or all) of a previously existing design as specified by the user. LL checks the validity and consistency of the links entered by the user for the purpose of forcing all or part of the line layout.

Another command which may be used is the ADD command. It allows the user to find the best circuit to add a terminal node to and determine the incremental cost of adding the node. ADD (and its existing counterpart DROP) should only be used to maintain an existing network. The DROP command allows the user to modify a layout by removing a node.

Report Generator and Cost

COST gives a brief summary report on the cost of the network on a concentrator-by-concentrator basis and is useful for obtaining very quick cost comparisons. For example, we might wish to compare two design strategies (one with and one without concentrators) or see the effect of reassigning a node to a different concentrator.

REP gives a detailed report on the cost and makeup of one or more circuits. A report on the entire network may be obtained. These reports will contain the trunks connecting the concentrators to the center and to one another. In addition to the cost of each circuit, these reports include a listing of the drops and interexchange lines in a format very similar to that of a telephone bill. Also included in the report is a summary of the loading on each circuit.

Reliability Analysis

TOPO allows the user to obtain an analysis of the availability of a given network in terms of the availability of its components. The analysis is presented via several figures of merit computed on a device-by-device basis or alternatively on a systemwide basis. Combinations of these approaches are also possible. Availability is defined as the fraction of time a given device is functional. The user can also specify redundancy factors either individually or on a systemwide basis. The model used for the purposes of reliability analysis is essentially a serial one, with parallel devices permitted for redundancy. Thus, the model requires that all devices on the path between the terminal and center be functional for communication to take place. If redundant devices are specified, any one of the redundant devices is sufficient to allow communication to take place.

The reliability analysis places somewhat more stringent constraints on the type of topology permitted than does the rest of TOPO. The REL analysis outputs five figures of merit characterizing the reliability of the network on a systemwide basis. The figures are average values. The first is the percentage of terminals communicating with the center. The second is the percentage of traffic able to communicate with the center. The third is the percentage of terminal pairs which are able to communicate. Fourth is the fraction of traffic able to communicate through the center. The last figure of merit is a distribution of the number of failed terminals, i.e., the fraction of time (on the average) that a given number of terminals will be unavailable.

Network Checkout

TOPO's CHECK command allows the user to execute a subsystem which will go through the network data base and perform several error checks. These checks are designed to identify many of the most common user errors which would cause difficulties if TOPO's design or analysis programs operated on them.

Availability

Available on the ARPANET.

Information Sources

This information was taken from Reference 24.

POD (PERFORMANCE ORIENTED DESIGN) TOOL

Introduction

POD (Performance Oriented Design) is a tool for modeling the performance of computer based systems at various stages in the system life cycle. Essentially, POD deals with time related and capacity related quantities such as throughputs, response times, bandwidths, utilization levels and component loadings. POD does not address factors such as tactical effectiveness, reliability, availability and logical correctness. However, POD can be used in conjunction with other software engineering and system design tools that do treat these factors.

Use of Performance Models

The basic function of any performance model is to determine the performance of a "target system" when it is operating in a "target environment." A POD model is a computer program that accepts as input a description of the target system and target environment. The output of the POD program is a set of numerical values that characterize the performance of the system in the environment. In the early phases of the system life cycle, inputs to POD performance models consist of estimated values, values specified in the documentation of system requirements and, in uome cases, a few known values determined by measuring related existing systems. As implementation proceeds, estimated values are continually refined and ultimately replaced by actual measured values.

Present Capabilities

The present version of POD requires the specification of hardware and software in terms which describe their interconnections and interactions. This is achieved through the POD System Description File which includes Configuration, Workload and Module Specifications. The Configuration Specification describes system hardware characteristics (disk and random access storage capacities, number of CPUs, device speeds) and interconnections. The Workload Specification provides the model with a definition of the external source(s) of jobs for the system to process. Job characteristics such as priority and arrival rate are supplied in this component of the system description of system software structure for processing the workload. Logical flow of control and resource utilization requirements are specified here. POD enables an analyst to enter an initial System Description File which may be refined as design and development proceed, increasing the detail and accuracy of the model. Proposed changes to an existing system or design may be evaluated by altering this file and then creating a new system model for analysis. System Description File input and update, system model generation and system analysis reporting are controlled interactively by the analyst.

Performance Oriented Design has recently implemented extensions to better model additional systems. They are also useful to more accurately describe a CPU. Different types of instruction groups and their associated execution rates may be described, enabling more accurate description of CPU utilization for different purposes. The Module Specification may now specify CPU utilization by a selected instruction group (logical, arithmetic, higher level language, etc.) offering more flexibility and accuracy in the model description.

In Reference 42 (January 1982) a recommendation is made for proposed FY82 tasking to BGS Systems. This task was expected to be completed no sooner than October 1982. Documentation describing these extensions to POD will be available in the future. This task is desirable to model an executive in a distributed multicomputer environment and would result in an updated POD to model the overhead associated with intercomputer communications. Appropriate hardware interconnections between multiple CPUs must be developed so that the model can be run and statistics gathered without resorting to multiple runs in an iterative procedure. In addition, the capability to run multiple execs and applications tasks concurrently would be desirable.

Special Features of POD

POD has a number of special features that have been developed specifically to support life cycle performance modeling of computer based systems. The most important of these features are:

1. A front end specifically tailored for modeling computer based systems
2. Facilities to support life cycle, rather than one-shot, modeling
3. An interactive modeling environment
4. Internal algorithms based on analytic queueing theory rather than discrete event simulation.

POD Front End

The key to discussing the POD front end is to describe the conceptual view of computer based systems that POD users are expected to have. To the POD user, hardware components are a set of "parameterized black boxes" whose internal workings are essentially invisible. In all these cases the internal engineering decisions that have led to a given set of parameter values are hidden from the POD user: the only visible properties of a hardware component are the parameters themselves. For the intended POD user, software differs from hardware in several extremely significant ways. First, software does not consist entirely of a set of "off the shelf" components. On the contrary, part of the job of the POD user is to design and implement software to perform specific functions. Thus, the highly variable and richly detailed internal structure of software is directly visible to the POD user. In addition to having a visible internal structure, another very important difference is that the estimated performance characteristics of individual software modules are likely to change significantly during design and implementation. Also, the actual structure of software is not known to a fine degree of granularity in the early stages of system development. That is, POD must support the coarse granularity representations of software that are provided in the initial stages of system design and must allow these representations to evolve smoothly into the more detailed representations that are generated subsequently during design and implementation. In the view of the POD user, the environment in which a computer based system operates is regarded as a set of external stimuli that causes software modules to be executed at various points in time.

POD Facilities For Life Cycle Modeling

In addition to a front end that is tailored for modeling computer based systems, POD provides special features to support "life cycle," as opposed to "one shot," modeling. That is, POD provides facilities that enable models to evolve over time and retain continuity with earlier versions, assumptions and estimates. One life cycle facility that is given above is the modular representation used for software. At times when detailed information regarding software structure is unavailable, the POD user can specify coarse granularity "aggregate" modules. Later in the life cycle, when refined structural information is available, an aggregate module can be decomposed into a series of more detailed modules. Another POD facility that has been specifically included to support life cycle modeling is the Project History Data Base (PHDB). The PHDB allows POD users to extract a variety of information from POD models, store this information in a data base and then prepare reports on trends, exception conditions and changes over time.

Interactive Modeling Environment

Another important feature of the POD modeling tool is its interactive mode of operation. The POD user is able to work at terminals, vary model parameters, request model evaluations and receive responses interactively.

Internal Algorithms Use Analytic Queueing Theory

POD does not use discrete event simulation to evaluate system performance. Instead, POD uses mathematical formulas derived from queueing theory.

Availability

POD for distributed systems is not available now.

Information Sources

This is taken from References 42 and 43.

SARA (SYSTEM ARCHITECTS' APPRENTICE)

UCLA's Methodology For Synthesis

The UCLA design methodology can be characterized as being requirement driven and supportive of self documenting design of modular, concurrent hardware and software systems. Given a set of requirements and assumptions about the environment, the designer conceives of a system to satisfy those requirements.

The fundamental rules which are recommended in this synthesis methodology are stated below.

1. Every synthesis task must be preceded by an explicit statement of intents.

2. Every significant aspect of intended behavior must be formalized and associated with a measure by which success of the design can be evaluated.

3. For any synthesis tasks, the environment of the system under design must be explicitly modeled. This serves to separate assumptions about an existing environment from the system under design. The validity of the final design is then guaranteed only under those assumptions.

4. Multilevel partition of a system under design must be allowable to manage complexity through a refinement process. Every subsystem can be treated recursively as a new synthesis problem.

5. Every synthesis must end by composing models of previously realized elements which have been defined and tested. The test environments of these elements must be consistent with the subsystem models which they are replacing and subsystem's environments.

6. It must be possible to create models which hold true without forcing implementation decisions until composition. It must be possible to introduce assumptions about implementation at any level of partition and have those assumptions hold through composition.

7. Models must be expressible in computer processible form to manage complexity.

8. The computer support system must be interactive.

9. It must be possible to separate structure from associated behavior in a synthesis model. Structure is meant to be a namespace for subsystems and their interconnections. Such a namespace is to aid the enforcement of modularity in a system design and it establishes the only points at which hypotheses about modular behavior can be tested by an external observer.

10. It should be possible to directly fabricate a system synthesized by the above methods.

Modeling Primitives

This section introduces a number of the modeling primitives used by SARA.

Module: A named module is intended to represent a structure whose internal fully nested structure is hidden from the outside. A module's only possible communication with the outside is through a socket. Other than through its socket, a module's name is known only to the structure within which it is nested.

Socket: A named socket is part of a module but the socket's name is known both inside and outside of its host module.

Interconnection: A named interconnection is a structure bridging two or more sockets.

Any of the above structural primitives at a given level of abstraction may be modeled in greater detail at a lower level by describing its internal structure in terms of the same module, socket and interconnection primitives. A behavioral model may be associated with a named module, a named socket or a named interconnection. The behavior of a module or an interconnection may be observed only through the structures' sockets according to the behavior associated with the socket.

Control Node: A named control node represents a step in a process being modeled. A controlled data processor or a simple delay may be associated with the control node.

Control Arc: A named control arc represents nonvolatile precedence relations between sets of nodes. Input control arcs enable the initiation of the respective steps represented by nodes when tokens are present. Output arcs pass a thread of control to one or more succeeding nodes. The arcs act as place holders for tokens.

Tokens: Tokens are status markers which are associated with control arcs to denote active threads of control. The token state of a control graph is initialized and then it can be changed by an abstract token machine consistent with conditions at the inputs and outputs of control nodes.

Input Control Logic: A logical relation among the input arcs to a node specifies the precedence conditions that must be satisfied by token states for the node to be initiated. Tokens from the initiating arcs which satisfy the input relations are absorbed from one of the initiating arc set governed by an OR relation in a manner established in the token machine and from all members of an initiating arc set governed by an AND relation.

Output Control Logic: A logical relation among the output arcs specifies which arcs have tokens placed upon them when a control node is terminated. When an exclusive OR output relation holds, a data processor interpretation must decide which arc receives a token. When an AND relation holds, all output arcs receive tokens.

Controlled Data Processor: A named controlled data processor, P1(NJ), represents a data transformer which is activated when its control node, NJ, is initiated. When the data transformation is completed, its control node, NJ, is terminated. An interpretation of the data transformation and delay occurring during the process is associated with P1 and, therefore, with NJ. Controlled data processors can properly communicate (Read or Write) via data arcs only with data sets, possibly through uncontrolled processors.

Uncontrolled Data Processors: A named uncontrolled data processor, UI, is used to represent data transformers which always provide a stated function of their inputs at their outputs. An interpretation of the data transformation and delay occurring during the process is associated with UI.

Data Sets: A named data set represents a passive collection of data. Data structure may be associated with a data set.

Data Arcs: A named data arc statically binds data processors and data sets. The data processor has Read or Write access to a data set, consistent with the data arc direction, whenever the processor needs it.

Modeling Domains

The Graph Model of Behavior (GMB) allows the modeling of various aspects of communication protocols in three modeling domains: control, data and interpretation. The control graph is a bilogic directed graph derived from the Graph Model of Computation (GMC) developed at UCLA and used by Postel to model protocols. The three domains provide flexibility in modeling the behavior of each component of a protocol. Modeling in the control domain often permits completely automatic analysis of a protocol. However, the control flow analysis may lead to an explosion of the number of states. Use of the data and interpretation domains makes available abstractions which significantly reduce the number of control states while providing enough information to permit semiautomatic verification. When neither of the above analyses is possible, the complete GMB provides a good environment for simulation experiments.

The Control Graph

The control graph consists of nodes which model events and arcs which model precedence relationships among events.

Each node has an input logic expression which dictates the conditions under which the node is to be initiated. An "OR" (+) in the input logic means a node may be initiated through any one of a set of designated arcs. An "AND" (*) in the input logic means that all arcs must pass control before the node is initiated. A node's input logic expression may be an arbitrary function of its input arcs using "AND" and "OR" operators.

Each node has an output logic expression which dictates the arcs through which the node passes control upon termination. An "OR" implies a decision to be made among designated arcs. An "AND" implies the passage of control through more than one arc. As is the case with input logic expressions, output logic expressions may be complex functions using the conjunctions (*) and disjunction (+) operators. Figure 23 illustrates the control primitives with their input and output logic expressions. It also shows a complex arc R, which is used whenever there are multiple source and destination nodes.

A control graph is a static description of possible control sequences. The control state of such a graph is represented by the distribution of tokens on control arcs. When a node is initiated the tokens which enabled it are absorbed. Upon termination of a node, tokens are created and placed on its output arcs as dictated by the output logic. The semantics of the control graph are carried out by underlying machinery: the token machine. This token machine performs state-to-state transformations dictated by the graph, starting from the initial token distribution, and terminating when no further transformations are possible.

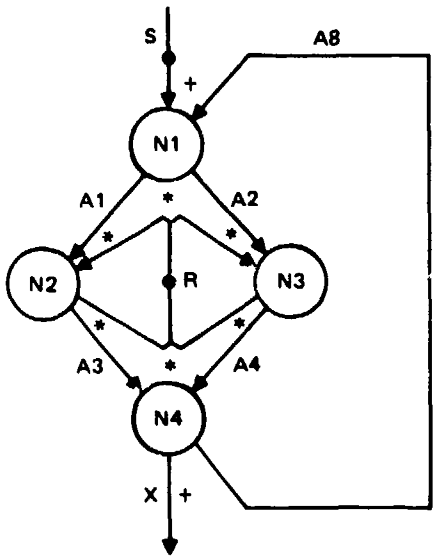


Figure 23. Control graph.

The Data Graph

The data graph consists of data sets, controlled processors, uncontrolled processors and data arcs. Data sets model static collections of data. Controlled and uncontrolled processors model data transformers. The initiation of controlled processors is dictated by the initiation of control nodes to which they are explicitly mapped. The termination of controlled processors causes the termination of their initiating control nodes. The initiation of uncontrolled processors is independent of the control graph: they are triggered by changes to any of their input data sets. Data arcs model data paths and access paths between data sets and processors. The semantics of all data graph primitives are carried out by the token machine.

Interpretation

Interpretation is used to define formats of data stored in data sets and transported through data arcs. It is also used to define transformations on data performed by data processors and to define the decisions needed to determine flow of control.

Data formats are specified in the form of PL/I declarations. Data transformations are specified in the form of algorithms written in PL/I and augmented with constructs used to effect delays in the flow of control and flow of data and to effect decisions in the flow of control.

Availability

SARA is available on the ARPANET.

Information Source

This information comes from References 27 and 28.

GRINDER

Introduction

The designer of a communications network is usually faced with many requirements that have complex interrelationships. As good design and evaluation techniques become available for various aspects of network design, the need arises for software that will provide convenient access to these techniques and permit easy and effective designer interaction with them. The GRINDER (Graphical Interactive Network Designer) software package was developed at Network Analysis Corporation (NAC) to provide such a capability.

The various modules in GRINDER can be described in this way: ACUT will do topological modifications to obtain a lower cost network given a traffic matrix, tariff and a starting topology under constraints on throughput and delay. It may add or delete capacity between nodes. GLAD designs the connections in a local access network given constraints such as the maximum number of concentrator ports available or line traffic capacity. A description of the nodes and traffic is required and the tariff selection needs to be specified. HIER designs the backbone network given a network configuration describing the host nodes, terminal nodes, backbone nodes and the traffic information. ROUT determines maximum throughput and optimal routes given the traffic information under a maximum end-to-end delay constraint.

The graphics routines are unavailable to the Navy at this time.

Network Editor

The Network Editor maintains a "network data base" which can be modified, viewed and saved. The data base consists of nodes, arcs and links and the properties which define them. It can be saved on disk and then read back into the Editor at a later time.

Nodes are connection points in the network. In some manner they usually initiate, terminate or move information through the network. They are labeled by a node identifier. Each node has an associated set of property values which may be modified by the user.

The program distinguishes between links and arcs. A link is defined as the entire path, with no intervening nodes, between a pair of nodes; an arc is defined as a particular path between two given nodes. Links can be referenced by the node pair alone. Arcs must have an identifying label. The links of the network are the connections between node pairs. Links are defined by their node pairs and also have properties which can be modified by the user. Arcs are links with labels. Arcs can be referenced at all times by their label or by the node pair if there are no other arcs existing between that node pair. Arcs, like nodes, have properties which can be modified by the user.

TRAFFIC EDITOR

The performance of a network depends on the data traffic that the nodes transmit and receive. The designer typically wants a network configuration that will give satisfactory performance under a specific traffic load and pattern. The Traffic Editor allows the user to specify a wide range of traffic patterns and loads.

There are three components to the data traffic load and pattern:

1. The Uniform Traffic Requirement is an amount of data traffic that is sent from all backbone nodes to all other backbone nodes.

2. The node to node Traffic matrix is used if there are special data traffic requirements between specified node pairs. Each entry in this matrix is added to the Uniform Requirement to come up with the total data traffic requirement between each node pair.

3. The point to point Traffic requirements.

The traffic editor will be responsible for most manipulation of traffic requirements. It will consist of a number of routines which can be accessed through the information structure described above.

The traffic editor will perform the following functions:

1. Read a triplet file from a disk file and store the backbone to backbone requirements in an NxN matrix, store the point to point requirements in transmit and receive locations by node and, if given the mapping of nodes to backbone nodes, collapse all point to point traffic requirements to the appropriate backbone nodes.

2. Write a triplet file of the backbone-backbone matrix.

3. Interact with the user to change or scale portions of the backbone matrix.

4. Produce random traffic patterns given a base requirement.

5. Produce uniform traffic requirement.

The data base necessary for the editor will consist of the following:

1. Number of nodes (NN)
2. Maximum number of nodes (MAXNN)
3. Vector of node labels (NL(NN))
4. Vector which maps nodes into matrix
5. Matrix of requirements (TR(MAXNN,MAXNN))
6. Code for a backbone node
7. Vector which maps all nodes to a backbone node

TARIFF EDITOR

The overall purpose of a tariff editor module is to give GRINDER users the ability to select tariffs for all line costing routines. The tariff editor will allow the user to select one to five different tariff structures from any of the available tariff modules. The structure will easily permit implementation of new tariffs. The user can cost out his/her network by one uniform tariff everywhere or by a unique tariff in each costing routine. The user can easily compare the cost of his/her network by two (or more) different tariffs.

Currently, there are two non-null tariff structures which can be used. These are: PLT - Piecewise Linear Tariff Structure and TLPK - TELPAK. PLT models such situations as simple cost per mile, bulk rate circuits with a known average utilization and current narrowband tariffs. TLPK is a speed dependent tariff structure. It models the situation in which TELPAKS are available at all points.

A TELPAK is an AT&T tariff that allows a user to purchase either 60 or :40 service equivalent lines at substantially reduced rates. The user must :ay for all 60 or 240 lines even if he/she requires fewer lines to meet .s/her needs. The TELPAK design problem is then to find the most economical arrangement of TELPAKs and private lines to meet a set of flow requirements.

In this editor, the user can invoke tariffs for GLAD, ROUT, HIER/CBA and HIER/LAL. Any routine that costs out links will be able to use any tariffs that were present.

Whenever a module that does costing is called, the user will be asked which tariff he/she selected applies (NOTE: GLAD users can indicate two choices for trunk tariffs and two choices for multipoint tariffs). A tariff executive program will then be called for all costing and it will know which tariff will be used and all of its parameter values.

ACUT

The programs contained in the ACT module of GRINDER are for the running of the Automated Cut Saturation Algorithm (ACUT). The module contains the routines necessary to automatically design a low cost packet-switched network. The networks generated will meet a set of user specified performance measures (i.e., throughput, delay, preserving 2-connectivity, etc.). The programs will report to the user the topological modifications made along with throughput and cost (the critical performance measures in the design phase).

The ACUT algorithm is essentially an "intelligent branch-exchange" algorithm. In simplest terms, the algorithm attempts to maintain the network's throughput performance between two bounds while reducing cost. At a particular design iteration, a topological modification is made. To maintain or bolster the throughput performance, capacity in the form of communication lines is added across the "saturated-cut set." To reduce cost, capacity is removed from areas in the network not in the saturated cut. After a topological modification is made, a routing analysis is invoked (obtaining new cost, throughput and delay performance measures) on the new topology. The algorithm then computes the new saturated cut-set and determines the best topological modifications to be made.

As a figure of merit, a network is said to be "undominated" if no other network costing less has a higher throughput. Alternatively, a network is said to be "dominated" if a cheaper costing network provides higher throughput performance.

The ACUT module is accessed through GRINDER in the ROUT module. It is not a proper set of subroutines but rather a program running in the background mode. Communications between GRINDER and ACUT are through transitory files.

The inputs to ACUT, which are specified via the ACT front-end program, consist of four basic parts: the network configuration including some general network properties, parameters specific to the ROUT program, the node to node traffic matrix and parameters specific to the ACUT programs. An initial network configuration is required by the ACUT programs.

The ROUT parameters define network performance criterion and constraints and tolerances which will affect the accuracy and running time of the ROUT programs in ACUT. Probably the most important parameter is the TARIF parameter that points to the tariff to be used in ACUT. The topological choices that ACUT will have will be determined by the tariff defined in the TARIFF EDITOR and used by the ROUT programs. In particular, the number of speed options defined in a tariff will determine what modifications are available.

The node to node traffic matrix is also required for input.

The ACUT parameters are used as constraints in the ACUT module. The ACUT parameters and their meanings are given below.

MAXML - MAXIMUM LINK MULTIPLICITY. In the course of adding links, ACUT may add capacity to existing links by either upgrading to a higher capacity option or adding another channel of the same capacity. This parameter is an upper-bound on the number of channels on one link.

MAXTH - MAXIMUM THROUGHPUT LEVEL, MINTH - MINIMUM THROUGHPUT LEVEL. Expressed as a percentage of the base throughput level, these parameters control the design actions ACUT takes. If the network throughput is below MINTH, ACUT will only add capacity to the network. If the throughput is above MAXTH, ACUT will only delete capacity. Between these levels, ACUT will attempt to both add and delete capacity in a given iteration.

DELMX - MAXIMUM OF DELETE CANDIDATES EXAMINED. During a delete operation ACUT will examine various delete possibilities. When a deletion is considered, tne routing optimization will be invoked. If the resulting network is dominated, the next best delete candidate is examined again by invoking the routing optimization, etc. This parameter controls the maximum number of deletion candidates examined and, therefore, the number of times the routing optimization is called at a particular design iteration. For small networks, this number can be large since the routing optimization runs faster for smaller networks. For larger networks, to speed up the design process, this number should be rather small.

Description Of ACUT Algorithm

The ACUT executive, upon determining that it has a starting topology and the appropriate parameters, will iteratively call on ACUT to provide topology modifications in the following manner:

1. Initialization. The routing program is called to determine the cost and throughput of the starting network.

2. Saturated Cut Determination. The cut determination algorithm is called and it divides the nodes into two components. A node will either have the component number of one component or the other, or it will be identified as part of a chain in the cutset.

3. If the current throughput is below the maximum throughput requirement, then capacity is added across the saturated cut. The throughput is not recalculated at this point.

4. If the current throughput is above the minimum throughput requirement, capacity, not across the cut, is deleted.

5. A check is then made for redundant cost. If the cost is not redundant, then it invokes the routing program to determine the flows and throughput. At this point a check for dominance is made.

6. If the resultant throughput is below the minimum throughput requirement or the network is repetitive or dominated and a capacity deletion has just occurred, then cancel the previous delete action and substitute another delete action for it, as described in the section concerning deletion of capacity and go to step 5. This procedure will be referred to in the future as "backtracking." If all available deletion options have been exhausted or the user imposed limit on deletion attempts has been reached, then attempt to implement the added capacity, if any, and go to step 5. Note that no deletion has occurred at this stage.

7. If the network's throughput is accepted and the network is nonrepetitive and undominated, then store the new network in the state vector. Remove from the state vector all networks which are now dominated by the new network. Return the add and/or delete actions to the ACUT executive.

8. If the conditions of step 7 are not satisfied then return no modifications to the ACUT executive.

GLAD

The GLAD module designs local access networks which can be described as follows:

Three levels of network hierarchy are involved. At the lowest level are known access positions generically called terminal sites. They require connection to known positions generically called centers, which form the highest level of the hierarchy. The centers may actually correspond to backbone sites which are not actual resource sites, but which provide appropriate interconnection. Intermediate between these levels are positions generically called concentrator sites. The concentrator sites are usually not precisely known. The program seeks to connect the terminals to the centers as efficiently as possible, placing concentrators at eligible sites where they are cost advantageous. The programs can handle the simpler situation where no concentrator sites are permitted and there are only two levels of hierarchy. All of the known and potential sites are collectively called the nodes of the network.

A local access network can contain one or more partitions, each partition consisting of a center node and its associated terminal nodes. Each partition is designed independently. Partitions are produced in the HIER module. The terminals are connected to a single concentrator and/or center by tree structures. Each concentrator is connected directly to a center. Other topology options, such as loops and dual homing, will be added in the future.

Lines attached to terminals will be called multipoint lines and lines between concentrators and centers will be called trunks to distinguish them.

The user can impose a large variety of constraints on the design, such as maximum number of concentrator ports available or line traffic capacity. The TARIF routines are used by GLAD to cost lines.

The design of the concentrator sites and the connecting links that form the network is developed in four distinct phases:

1. Partition of the terminal sites into sets associated with particular centers; this is accomplished in the HIER module
2. Formation of the center-of-mass (COM) clusters of terminals
3. Selection of concentrator sites and sizes
4. Design of the interconnecting line layouts

The programs are set to handle 500 nodes. The arrays can be enlarged to accommodate larger networks. The amount of storage needed is approximately proportional to the number of nodes. Two sets of input data are required:

1. A detailed description of all the nodes and traffic involved in the network
2. The network design constraints, program execution options, tariff selections and other required program parameters

HIER

The programs contained in the HIER module of GRINDER allow the user to design and evaluate a 2-level hierarchical network architecture. This type of configuration is typically used for a very large distributed network application (up to several hundred host computers and several thousand terminals). The two levels can be viewed as separate networks. The common structure of large distributed networks is a multilevel hierarchical structure with a backbone network at the higher level and local access networks at the lower levels. Backbone and local access nets may be further subdivided into hierarchical sublevels. The backbone network is characterized by distributed traffic requirements and is generally implemented using the packet-switching technology. Local access networks, on the other hand, have a centralized traffic pattern (all the traffic is to and from the gateway backbone node) and are, therefore, implemented with centralized techniques such as multiplexing, concentrating and polling.

The HIER module in GRINDER manages various programs that allow a user to construct and analyze such a network configuration. In particular, he/she can do the following:

1. Evaluate a manually created network
2. Automatically choose a new set of backbone nodes
3. Automatically assign the low level host and terminals to the backbone nodes
4. Prepare the backbone network for analysis by ROUT
5. Optimize the local access network layouts (GLAD)
6. Derive a local point to point traffic requirements file from a backbone traffic matrix
7. Evaluate and cost the three components to such a network configuration; these are: the local access network, the backbone switches and the backbone network cost

Each program in HIER can be viewed as a self-contained entity that performs a particular task. There is no strict limitation on the order in which each module is invoked. However, some modules expect the network configuration to be in a particular state to perform its computation. There are no performance evaluation modules in HIER. They are basically design and cost oriented.

The input to the HIER module consists of the network configuration, host nodes, terminal nodes, backbone nodes and the links that interconnect them. There are several properties of the nodes that are particularly important.

The editing/design modules result in a network revision. These modules add/delete nodes or links, assign and reassign low level nodes to backbone nodes and generally reconfigure a network. The DASG Module deletes low level links to one or all backbone nodes. It does not affect any links between two backbone nodes. The ASG assigns each low level node to the nearest backbone node, by adding a link to the backbone node. It also assigns capacity to the added link. The DUHM Module performs the basic operation as ASG. The exception here is that a nonbackbone node that has been assigned dual homing capabilities is linked to a second nearest backbone. The CBC Module selects backbone nodes using a clustering technique called CPART. Previous backbone nodes are removed from the network.

The CBA Module is very similar in structure to CBC. CBA selects backbone nodes using an adding technique called APART. Previous backbone nodes are removed from the network. GLAD is a special package that designs the local access networks revolving around one or more backbone nodes. After Glad execution, HIER gets concentrator locations, if any were chosen, followed by all new local access links. Cost, length, capacity and flow on each link are also returned. HIER reports total cost of all local access links. The CSLO module chooses Satellite-IMP locations given a set of backbone nodes.

The traffic manipulation modules perform the task of handling the large amounts of traffic information of a large and complex network. The traffic information, in the form of triples, is found in auxiliary files. These traffic triples consist of two node labels and the traffic from the first node label to the second node label. The user is required to have initialized the backbone traffic matrix via the TRAFFIC EDITOR before entering HIER.

The Statistical Analysis modules offer the user short, concise reports of network relationships. Summaries of cost, traffic, devices and hardware associations become available through these modules. The BNA module reports assignments to one or more backbone nodes. It reports number of hosts and concentrators, terminal nodes, terminal devices, total hosts per concentrator transmit traffic, total host per concentrator receive traffic, total terminal transmit traffic, total terminal receive traffic and total traffic at the specified backbone.

The LAL module calculates the total cost of local access lines in the network. It uses a preset tariff. The LAH module reports the total local access hardware cost. By using node hardware type, this routine summarizes all hardware costs for nonbackbone nodes. The CST module costs out one or more given backbone switches.

ROUT

The programs contained in the ROUT module of GRINDER are quite powerful in designing and evaluating the performance of a packet switched network. A designer typically wants to obtain a low cost configuration that meets specified performance measures (i.e., throughput, delay, etc.). A network data base is created via the Network Editor. ROUT assumes that the nodes are packet switches and the links are the data channels connecting them.

The analysis determines the maximum throughput and the optimal routes in a packet switched network for a given source to destination traffic pattern, under the condition that the end to end average delay T over all node pairs be smaller than a specified constraint. The optimization of the routes is obtained using nonlinear network flow techniques. The procedure iteratively finds a sequence of improved alternate routes. Each iteration finds new routes between some node pairs and optimally distributes the traffic between the new routes and the existing routes. The optimization terminates when further significant improvements in throughput can't be found or at the user's direction. All requirements are scaled by the same factor during the throughput optimization process. The basic optimization method is the Extremal Flow (EF) Algorithm.

The inputs to the ROUT analysis consist of three basic parts: the network configuration including some general network properties, parameters specific to the ROUT module and the node-to-node traffic matrix.

The configuration is created and maintained by the NETWORK EDITOR. The ROUT analysis will only use those nodes in the NETWORK EDITOR that have hardware type equal to

BKBN - for regular Backbone nodes
GRST - for Ground Station nodes
SATL - for Satellite nodes

The current limitation on the maximum number of nodes is 65. The ROUT analysis will fill in the following NETWORK EDITOR link properties for inspection by the user:

1. Link cost ($/mo.)
2. Link length (miles)
3. The flow in both directions, if it is a link, or flow in directed direction, if it is an arc

The ROUT parameters define network performance criteria and constraints, the tariff that will be used for costing purposes and tolerances which will affect the accuracy and running time of the analysis. The ROUT parameters are:

ITMAX - Desired number of Iterations. This parameter can control the type of routing strategy that will be used. If ITMAX=1, the routing strategy will be the initial min-hop routing. If ITMAX>1, alternate routing will be performed. The first routing will always be min-hop routing but subsequent iterations find performance improving alternate routes between some node pairs.

PKLEN - Average. Packet Length (KBITS). The average size of the packet is listed in kilobits, including the packet header.

DELAY - Average. Packet Delay (SEC). The constraint on the end to end average packet delay for all node pairs is in seconds. The program will maximize the throughput while maintaining this constraint.

PROVR - Protocol Overhead (FRACTION). This parameter models the overhead associated with the particular protocol used. This value could reflect ACKS, RFNMS, packet header length, etc., all the overhead that is proportional to the data traffic. The program multiplies all the traffic requirements by 1+PROVR to get the actual traffic requirement (data + overhead) that the network must carry. The throughput results reported by the analysis will be only the actual data traffic. However, the flows reported on the links will include this overhead.

RUOVR - Routing Overhead (FRACTION). This parameter models the overhead traffic the network must accommodate because of routing and control information. This could reflect routing table updates, network control information, etc. RUOVR is, however, speed dependent. The program reduces each line capacity by this fraction.

BPRCS - Backbone Nodal Processing Time (SEC). This parameter specifies the time a regular backbone node takes to process the average size packet (assuming zero-load).

GPRCS - Ground Station Nodal Processing Time (SEC). This parameter is the time a ground station node takes to process the average size packet (assuming zero-load).

SPRCS - Satellite Nodal Processing Time (SEC). This parameter is the time a satellite node takes to process the average size packet (assuming zero-load).

TPROP - Terrestrial Line Propagation Delay (SEC/MILE). This parameter is used to compute the transmission delay on a terrestrial line.

SPROP - Satellite Line Propagation Delay. This parameter is used to compute the transmission delay on a line between two satellite nodes.

MPRPR - Mixed-Media Line Propagation Delay (for Rout Analysis). This parameter is used to compute the transmission delay on a mixed-media line. This is the delay that will be used during the flow assignment portion of the algorithm.

MPRPD - Mixed-Media Line Propagation Delay (for Delay Analysis). This parameter is used to compute the transmission delay on a mixed media line. This is the delay that will be used at the final delay evaluation.

THACC - Relative Throughput Accuracy. This parameter sets a stopping condition for the analysis. If the relative throughput increase at a particular iteration is no greater than THACC, the analysis will stop.

TIACC - Time Delay Accuracy (SEC). The analysis program will attempt to maintain the delay constraint DELAY ± TIACC. The smaller the value of TIACC, the longer the program will spend adjusting the throughput level to match the delay constraint.

RTBSW - Save data for RTB; Yes or No. A flag that (if on) will save various routing data for the RTB post-processor. This flag is currently ignored as the RTB post-processor is not implemented.

TARIF - Tariff Option (selection Label). This points to the tariff slot defined in the TARIFF EDITOR. This is the tariff that will be applied to cost the network. The tariff should have been initialized before entering the ROUT module.

The performance of a packet switched network depends on the data traffic that the nodes transmit and receive. The designer typically wants a network configuration that will give satisfactory performance under a specific traffic load and pattern. The backbone node to node traffic matrix is created and maintained by the TRAFFIC EDITOR. It should have been initialized by the user before entering the ROUT module.

The tariff should have been initialized before entering the ROUT module. The backbone node to node traffic matrix is created and maintained by the Traffic Editor. It should have been initialized by the user before entering the ROUT module.

After the user has set up the values of the ROUT parameters, node to node traffic matrix and the tariff, he/she is ready to run the analysis. These values will remain constant throughout a session except when the user changes them. The "R" command is issued and the program responds with "Computations Proceeding." Next, the network cost per month will be calculated, broken up into total cost, line cost, hardware cost and per node hardware cost. Also, each link's multiplicity will be determined. The program will next calculate the base traffic requirement. It is this amount of traffic that the network should satisfy. It is equivalent to the sum of all the entries in the node to node traffic matrix, plus the uniform requirement. The program will then compute and print out for each iteration the maximum throughput and the delay constraint. The first iteration will be min-hop routing and for subsequent iterations the program will attempt to deviate traffic flow from highly utilized paths to underutilized paths. When the desired number of iterations is reached, the optimization is stopped. The throughput is then calculated more precisely using an exact muitiserver delay formula. This computation is too time consuming to be used during the optimization, so it is only done at the final iteration. However, when all the links have multiplicity of one, this exact delay expression reduces to the approximate delay expression used in the optimization. Therefore, there will be a difference in the maximum throughput only if there are links with multiplicity greater than one. When the user has requested no more optimization iterations, the program will calculate some final statistics. The program will compute the delay (using the exact delay expression) at which the base traffic requirement can be satisfied (if it can be satisfied at all).

There are commands which allow the user to look at various properties of each link. These include capacity, traffic flow, channel multiplicity, channel length in miles and cost. Another command will display the throughput-delay table. After each analysis, a table is generated that shows the average packet delay for various levels of throughput. The throughput at saturation (infinite delay) is divided into 50 equal increments and the delay is evaluated at each increment using the optimized flow pattern.

Post processors are modules that are generally called after the execution of the ROUT analysis. Because they use many of the data structures that the analysis routines create, an analysis should be run before any post processor is invoked. The post processors provide more detailed information on delay, flow and structural properties and characteristics of the topology under consideration.

The END post processor evaluates and generates end to end delay statistics. The user can obtain end to end delay between any and all nodes pairs, average end to end delay (averaged over all node pairs), maximum end to end delay, distribution of end to end delays, average and maximum round trip delays, distribution of round trip delays and routes between any and all node pairs on the min delay path. The delay calculation in END is performed assuming three classes (at most) of traffic. In case of multipath routing the delay is evaluated along the lowest delay route. The END parameters allow the user to specify his/her priority structure, the level of throughput at which the delay is to be evaluated and the average packet size to be used.

The HOP post processor evaluates and generates min-hop distance statistics. The evaluation is strictly topological. Traffic flow, capacity and length of the lines are ignored. It also allows the evaluation of candidate links (links that might be added to the configuration to improve min-hop paths). The HOP post processor gives an evaluation of the current configuration and then, if desired, evaluates that configuration with each candidate link in the topology. In particular, the user can obtain min-hop distance for any and all node pairs, average min-hop distance, maximum min-hop distance, distribution of min-hop distances, routes between any and all node pairs on the min-hop path and assistance in deciding what links to add to reduce the maximum and ıverage min-hop distance.

The CAND submodule allows convenient and quick evaluation of a list of links that could be added to reduce the maximum and average min-hop distance. The user creates a list of candidates links which is then evaluated. The evaluation considers the current configuration augmented by each candidate link. In particular, the user can obtain the cost of each candidate link, the maximum and average min-hop distance for the configuration augmented with each candidate link and the reduction per thousand dollars of the maximum and average min-hop distance. The current limitation on the size of the list is 20 links.

The CUT post processor assists the user in identifying traffic bottlenecks in the network. It is based on the concept of a "cut-set." A good way to increase the network's capacity is to increase line capacity in the cut or add additional lines to the cut. The CUT module isolates the cut-set for the user, but does not decide how to increase the capacity of the cut and, therefore, the capacity of the network. The CUT module simply determines the cut-set.

Mixed Media Networks

Mixed media networks are networks with both terrestrial and satellite links. Two subproblems in particular make this general problem different from the design of a "pure" terrestrial network. These particular subproblems involve modeling satellite links and optimizing the number and location of satellite ground stations.

The selection of the most effective architecture (i.e., number of levels and type of access at each level) is the first task usually faced in the design of a large network. After the architecture is selected, network design then determines a topology that, in some sense, minimizes line and nodal processor costs. For the general case where a network can mix terrestrial and satellite transmission media, this task can be broken down into the following subproblems:

1. Preliminary clustering of user installations
2. Selection of backbone nodal processor locations
3 Local access design
4. Modeling of satellite transmission media
5. Selection of satellite switch locations
6. Terrestrial subnet design

Of the six subproblems listed above, 1, 2, 3, and 6, are common both to networks which employ only terrestrial transmission media and to networks which employ mixed media. The remaining two are totally irrelevant for pure terrestrial networks. They represent the additional design effort required to deal with the satellite segment of a mixed media network.

In representing the satellite media by appropriate models, the following assumptions are made:

1. The satellite is a linear, frequency translating repeater
2. Both uplink and downlink bandwidth allocation are each W Hz
3. The uplink and downlink carrier to noise ratios are such that the satellite transmission media are downlink limited and can support a total downlink data rate of C bps in a non-random access mode of operation
4. The number and locations of the satellite ground stations are specified.

With these assumptions the type of model used to represent the satellite media depends upon the satellite access scheme employed. In enhancing existing design tools to deal with mixed media, the following three types of satellite access schemes have been considered: point to point, channel division multiple access and random or reservation multiple access. (The reservation multiple access is a form of what is commonly called "Demand Assignment Multiple Access".)

Availability

GRINDER is available on the ARPANET on the ISIA computer using the TENEX operating system.

Information Source

This information was obtained from Reference 44.

CHAPTER 6

SUGGESTED APPROACH FOR NETWORK DESIGN TOOL

THE EHF NETWORK DESIGN PROBLEM

The EHF network design problem can be decomposed into the following elements:

1. Architecture. This element defines all of the network related functions and the supporting protocols. It places these functions and protocols into a structure following the ISO-OSI reference model.

2. Service Definition. This element takes the given user requirements and defines the network services that will support those requirements.

3. Resource Design. This element performs the design tradeoff on the physical elements of the system including the satellite antenna coverage schemes, time and frequency channelization for the uplink and downlink and switching designs.

4. Resource Allocation. This element defines methods for sharing the resources (communication channels) among users to provide the desired services.

5. Protocol Designs. This element defines the functions and design of the protocols required to implement the network functions. The various protocols include:

   a. Service (session) setup and teardown

   b. Channel access

   c. Intersatellite routing

   d. Intersatellite flow control

   e. Service preemption

   f. Network control transfer

   g. Dynamic network maintenance

      (1) Dynamic network topology

      (2) Dynamic channel effects

6. Data Base Design. This element defines the data bases at the terminal nodes and on board the satellite that are required to support the network protocols.

7. Protocol Validation. This element verifies that the specified protocols are complete and are logically correc' .

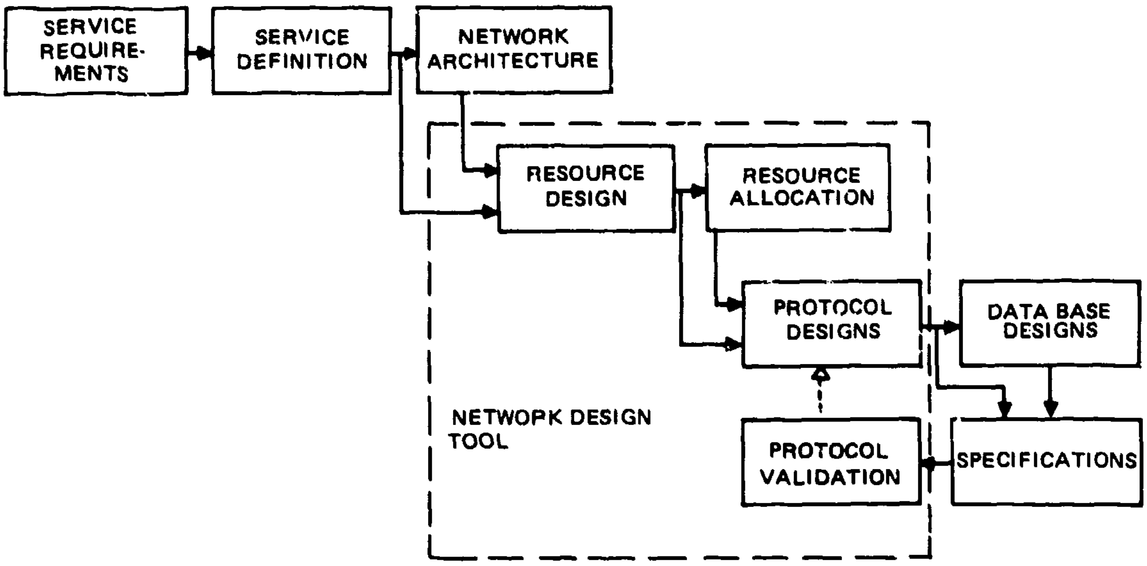


Figure 24. Network design tool applications.

Figure 24 shows the relationship between these design elements and shows which elements may be supported by the network design tool.

NETWORK DESIGN TOOL APPROACHES

The existing design tools described in Chapter 5 are useful for networks with fixed topology and unjammed channels. Most of the techniques apply only to point-to-point designs and do not apply to broadcast nets. The most well developed analytical methods are limited to topological designs and do not aid in the design of protocols themselves.

The following sections outline a network design tool for protocol design and network performance characterization. It uses several of the approaches listed in Table 3. These include queueing models, graph theory, finite state machine models and discrete event simulation.

Resource Design

The resource design problem requires a top level model of the EHF system including:

1. Uplink and downlink channelization
2. Service (network) requirements
3. Antenna configuration and coverage
4. Antenna beam-to-receiver switching scheme

A question is the availability of a channel (uplink and downlink) for a service request from a particular geographic area. The design parameters are the number of uplink frequency channels, the number of frequency channel groupings (that is, independently switchable entities) and assignment of uplink frequency channels to uplink antenna beams.

Under certain simplifying assumptions this design problem can be approached analytically, as in Reference 34. Simulation methods are required when the traffic statistics are non-Poisson or when satellite motion is considered. A hybrid approach may be used to include the dynamics of actual antenna patterns and specific user distributions over the earth.

Figure 25 illustrates a possible approach. User distribution and antenna coverage is determined by stored models. This information is combined with the given uplink and downlink channelization specified by the input data. At this point, analytical or simulation methods may be used to determine the channel availability and maximum channel capacity. Time varying characteristics may also be accommodated within the model, as shown in the figure.

Resource Allocation

The resource allocation problem occurs on (at least) two levels in the EHF system. First, there is the allocation, by the on-board resource controller, of uplink and downlink resources to a particular service. A service may be a network, so that within a particular service, a net controller may perform the channel access function among the participants in that service.

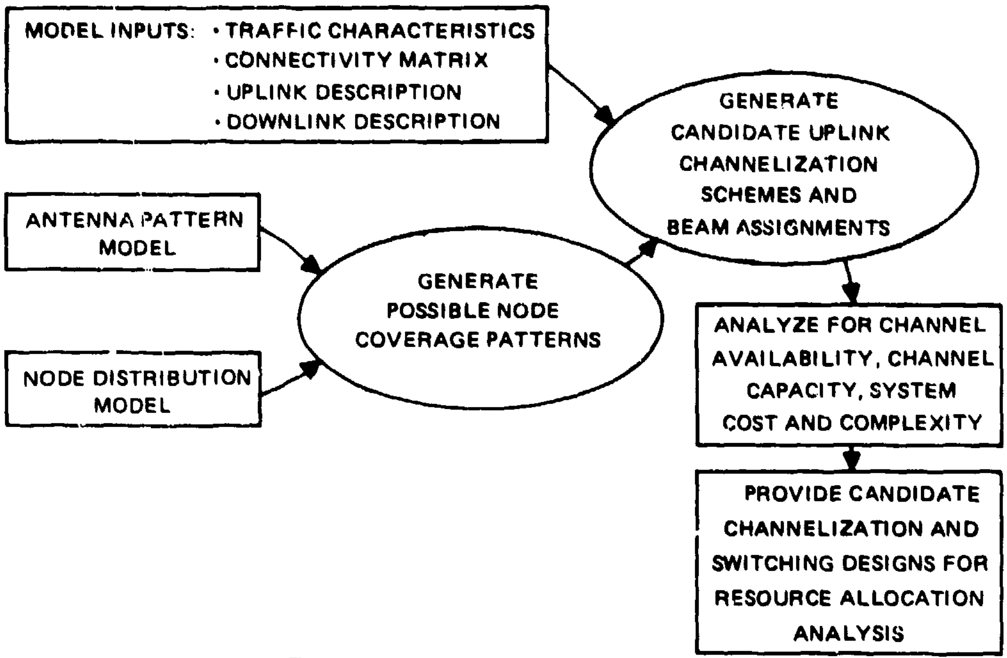

MODEL INPUTS: • TRAFFIC CHARACTERISTICS
• CONNECTIVITY MATRIX
• UPLINK DESCRIPTION
• DOWNLINK DESCRIPTION
GENERATE CANDIDATE UPLINK CHANNELIZATION SCHEMES AND BEAM ASSIGNMENTS
ANTENNA PATTERN MODEL
GENERATE POSSIBLE NODE COVERAGE PATTERNS
NODE DISTRIBUTION MODEL
ANALYZE FOR CHANNEL AVAILABILITY, CHANNEL CAPACITY, SYSTEM COST AND COMPLEXITY
PROVIDE CANDIDATE CHANNELIZATION AND SWITCHING DESIGNS FOR RESOURCE ALLOCATION ANALYSIS

Figure 25. Resource design.

The model parameters for the satellite resource allocation problem are determined by the resource design. The resource allocation analysis is a more detailed analysis that includes blocking probabilities, queue lengths and throughput rates for a given uplink and downlink configuration. The system can be described by a multiserver queueing model in cascade with a traffic multiplier to account for the multibeam broadcast downlink (Figure 26). This model can be characterized analytically for Poisson traffic statistics (arrival rates). If the traffic statistics are non-Poisson, or if the statistics are time varying, a simulation method must be used.

The service channel, once assigned by the satellite resource controller, has well known characteristics that remain reasonably constant. The EHF system supports a wide variety of access schemes including TDMA, polling, adaptive polling and DAMA. It supports both in-band and out-of-band control. There is no general technique for analyzing the performance of all these schemes. However, the use of queueing models appears feasible since the channel may be modeled as a single server queue in tandem with a service multiplier (to account for the multiple repetitions required for downlink broadcast coverage) and another queue (see Figure 27). Again, if the traffic statistics are non-Poisson, or if the statistics and/or traffic matrix are time varying, a simulation method must be used.

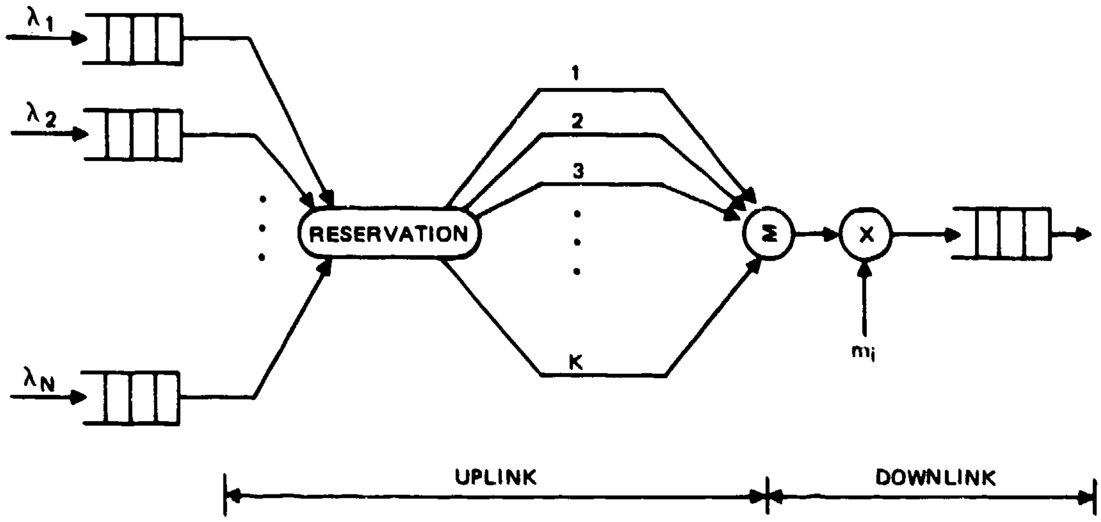


Figure 26. Channel access model for the satellite allocation problem.

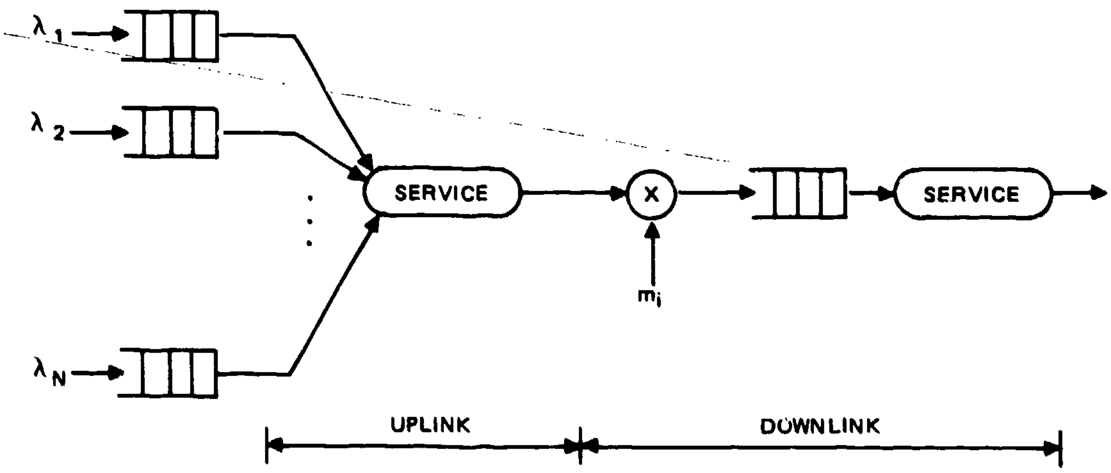


Figure 27. Channel access model.

Protocol Design

The designs of network protocols are usually done in an ad hoc fashion; however, the network design tool can be used as a design aid, if not the primary design method. For example, the network design tool may store descriptions of the protocols, as finite state machine models and/or sequence diagrams, that could be easily modified. This tool also could be used easily to expand the protocol descriptions to include error recovery techniques and timeout procedures. The tool also could perform worst case timing analysis for a protocol to show how long a cycle of events takes. Finally, it could provide a precise specification of a protocol to the designer for the protocol validation procedure.

Performance validation of most protocols will require simulation of the protocol. This may be done in an isolated fashion or it may be done in a structured, modular way as was described in Chapter 3 for the VANS system. This is even more attractive if the simulation model is structured to follow the ISO-OSI reference model. Using this approach, protocols may be analyzed along with known input data or together with another protocol which provides the input. A simulation tool may be designed to accommodate a general set of protocols, a subset of which may be selected and setup according to a user defined data base.

Protocol Validation

Protocol validation can be performed in an automated fashion using finite state machine models of the protocols. There is a limitation on this approach due to the number of states that can be accommodated, so the protocol models must be fairly simple. In addition, at present only the empty channel case (that is, pending actions are not allowed to queue up and actions are performed instantaneously) is accommodated. Within these restrictions, the validation procedure provides design rules to ensure valid protocols and checks for deadlock conditions and infinite loops.

NETWORK DESIGN TOOL OUTLINE

The architecture of the network design tool is outlined in Figure 28. The tool has three distinct elements, connected by an interactive executive. The elements are:

1. A system simulator, consisting of a network editor, a channel editcr and a simulator
2. A resource allocation designer
3. A protocol designer, consisting of the protocol design tool and the protocol validation module

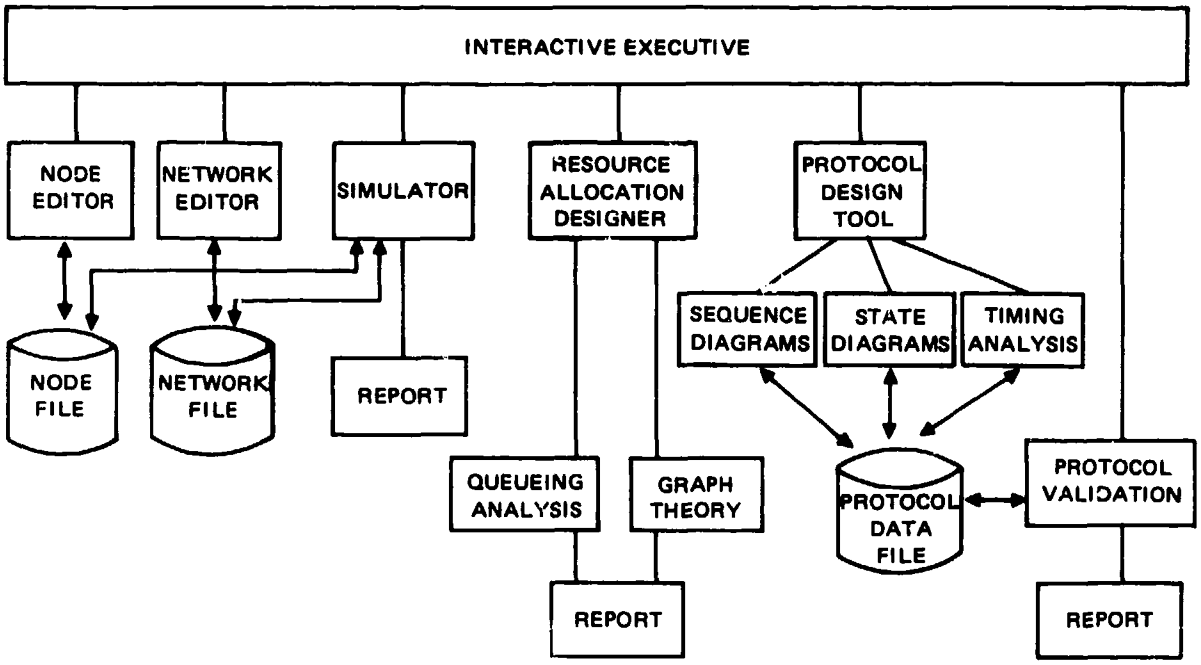


Figure 28. Network design tool architecture.

System Simulator

As described previously, the system simulator uses a modular approach where net activities are divided into distinct protocol areas. Interfaces between these areas us. a well defined interface standard so that different protocols may be easily exchanged. A protocol may be anlyzed in isolation by substituting a dummy data base at the protocol's input interface.

The designer defines the simulation by using the node and network editors. The node editor defines the protocol as viewed by a node for the data link, network and transport layers. Nodes can be defined at several levels (host, switching, backbone). The network editor defines the actual physical channel characteristics such as antenna coverage, channel data rate and time and frequency channelization. It also defines the desired network properties such as access technique, switching, connectivity and traffic characteristics. It may include dynamic modeling characteristics such as jamming, node movement, satellite movement and node destruction.

The simulator uses the files created by the editors to construct the simulation model and execute the simulation. It produces an output report as specified by the operator. The output can include steady state and transient results.

An advantage of this approach is that it can lead directly to a testbed implementation and then to a feasibility model, all following the same layered architecture.

Resource Allocation Designer

The resource allocation designer uses analytical techniques to perform topological optimizations and link designs. It uses the techniques discussed in Chapter 3 to optimize point to point network layouts and queueing analysis to design channelization schemes for satellite networks (see Reference 34). It also performs the analysis of multiple access schemes that fit within the constraints of the queueing models as outlined in the section on resource allocation.

Protocol Design Tool

The protocol design tool, as discussed earlier, operates as an aid to the designer. Since even simple protocols often become very complicated to describe, this tool provides a way of maintaining, modifying and reporting the designs. It can also guide the designer by checking for valid designs following the rules provided by Tamaki.[38] The protocol design tool will transform sequence diagrams (action-response versus time descriptions) into finite state machine models and vice versa. An automated protocol validation procedure will be available to analyze the finite state machine models for deadlocks and infinite loops.[38]

Network Testbed Approach

The network testbed is an approach that appears attractive for network feasibility testing in cases where a single topological hierarchy is involved and channel effects, such as jamming and platform motion, are not too complex. The approach is to create essentially a local net consisting of a number of general purpose processors tied to a high speed bus (Figure 29). Each node is emulated by a microcomputer, and a minicomputer is used as the central controller (if required). The minicomputer may also implement the channel model by intercepting all messages and performing processing to represent delay, message loss or degradation. The minicomputer also handles all program loading functions to minimize the cost of the microcomputers.

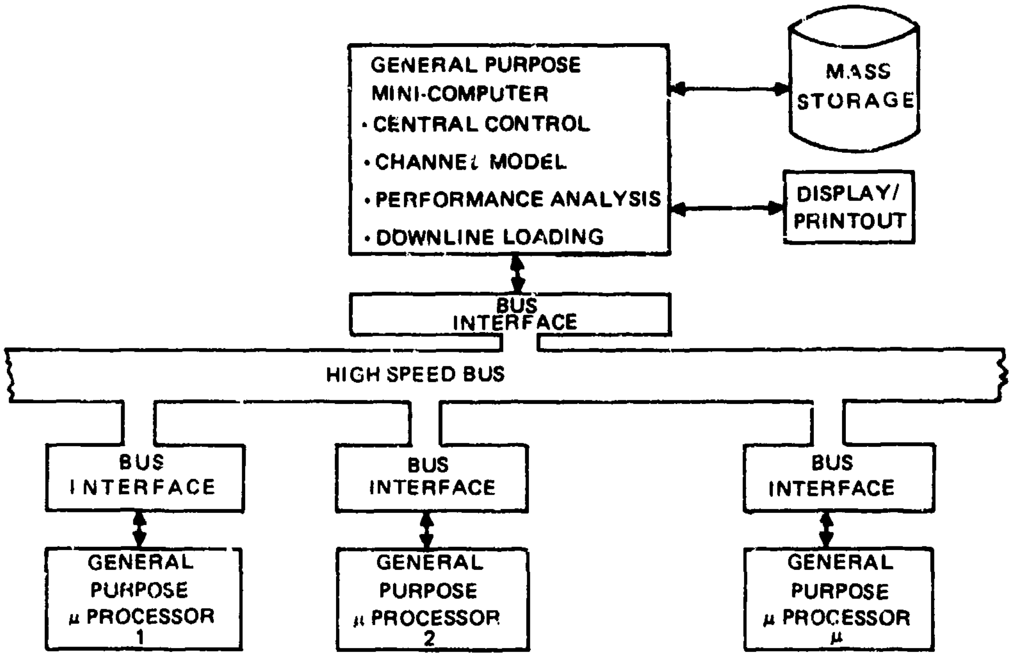


Figure 29. Network testbed.

REFERENCES AND BIBLIOGRAPHY

1. I. M. Soi and K. K. Aggarwal, A Review of Computer-Communication Network Classification Schemes, IEEE Communications Magazine, pp 24-32, March 1981.

2. H. Zimmermann, OSI Reference Model - The ISO Model of Architecture for Open System Interconnection, IEEE Transactions on Communications, Vol. COM-28, No. 4, pp 425-432, April 1980.

3. Roy D. Rosner and Ben Springer, Circuit and Packet Switching, Computer Networks, pp 7-26, 1976.

4. Fouad A. Tobagi, Multiaccess Protocols in Packet Communication Systems, IEEE Transactions on Communications, Vol. COM-28, No. 4, pp 468-488, April 1980.

5. H. Lee, On Design of the Least Vulnerable Networks of Minimum Delay, Department of Electrical Engineering and Computer Science, Northwestern University.

6. L. Kleinrock, Queueing Theory, Vol. I, John Wiley and Sons, New York, 1975.

7. L. Kleinrock, Queueing Theory, Vol. II, John Wiley and Sons, New York, 1976.

8. L. Kleinrock, Communication Nets, McGraw-Hill, New York, 1964.

9. Martin G. Kienzie and K. C. Sevcik, Survey of Analytic Queueing Network Models of Computer Systems, 1979 Conference on Simulation, Measurement and Modeling of Computer Systems.

10. L. Fratta and U. Montanari, Analytical Techniques for Computers Networks Analysis and Design, Computer Architectures and Networks, North-Holland Publishing Company, Amsterdam, 1974.

11. Gregor V. Bochmann, Finite State Description of Communication Protocols, Computer Networks, North-Holland Publishing Company, Amsterdam, Vol. 2, 1978.

12. G. Michael Schneider, The VANS System, Proceedings of COMPCON Computer Communications Networks, September 5-8, 1978, Washington, D. C.

13. G. Michael Schneider, A Modeling Package for Simulation of Computer Networks, Simulation, December 1978.

14. G. Michael Schneider, A Structural Approach to Computer Network Simulation, Computer Science Department, University of Minnesota, Technical Report 75-20, December 1975.

15. J. C. Browne, K. M. Chandy, R. M. Brown, T. W. Keller and D. Towsley, Hierarchical Techniques for Development of Realistic Models of Complex Computer Systems, Computer Architectures and Networks, North-Holland Publishing Company, Amsterdam, 1974.

16. Alf Hansen, The EIN Network Simulation, Computer Networks and Simulation, North-Holland Publishing Company, New York, 1978.

17. Antero Remes, Simulation Techniques in Network Design, Computer Networks and Simulation, North-Holland Publishing Company, New York, 1978.

18. John M. McQuillan and Ira Richer, A New Network Simulation Technique, Computer Networks and Simulation, North-Holland Publishing Company, New York, 1978.

19. J. Kent Peacock, J. W. Wong and Eric Manning, A Distributed Approach to Queueing Network Simulation, 1979 Winter Simulation Conference, December 3-5, 1979.

20. K. M. Chandy, Victor Holmes and J. Misra, Distributed Simulation of Networks, Computer Networks, Vol. 3, 1979.

21. J. Kent Peacock, J. W. Wong and Eric G. Manning, Distributed Simulation Using a Network of Processors, Computer Networks, Vol. 3, 1979.

22. Fouad A. Tobagi, Mario Gerla, Richard W. Peebles and Eric G. Manning, Modeling and Measurement Techniques in Packet Communication Networks, Proceedings of the IEEE, Vol. 66, No. 11, November 1978.

23. H. Gomaa, A. Hybrid Simulation/Regression Modeling Approach for Evaluating Multiprogramming Computer Systems, Computer Performance, edited by K. M. Chandy and M. Reiser, North-Holland Publishing Company, New York, 1977.

24. Modular Interactive Network Design (MIND) User's Guide, Version 5.1, Network Analysis Corporation.

25. Howard Cravis, Communications Network Analysis, Lexington Books, Lexington, MA, 1981.

26. J. Schwandt, An Approach to Use Evaluation Nets for the Performance Evaluation of Transaction-Oriented Business Computer Systems, Computer Performance, edited by K. M. Chandy and M. Reiser, North-Holland Publishing Company, New York, 1977.

27. R. Razouk and G. Estrin, Validation of the X.21 Interface Specification Using SARA, NBS Trends and Applications Symposium, May 29, 1980.

28. Gerald Estrin, A Methodology for Design of Digital Systems - Supported by SARA at the Age of One, National Computer Conference, 1978.

29. Rami R. Razouk, Computer-Aided Design and Evaluation of Digital Computer Systems, February 1981, UCLA Computer Science Department, Report No. CSD-810205.

30. M. Parent and I. R. I. A. Laboria, Graphical Models and the LAM Hardware Discrete Event Simulator, Computer Performance, edited by K. M. Chandy and M. Reiser, North-Holland Publishing Company, New York, 1977.

31. Symposium on Modeling and Analysis of Data Networks, National Science Foundation, March 1976.

32. E. Gelenbe and I. Mitrani, Analysis and Synthesis of Computer Systems, Academic Press, New York, 1980.

33. G. J. Foschini, On Heavy Traffic Diffusion Analysis and Dynamic Routing in Packet Switched Networks, Computer Performance, edited by K. M. Chandy and M. Reiser, North-Holland Publishing Company, New York, 1977.

34. Izhak Rubin, EHF Network Analysis Model, Final Report for NOSC Contract No. N66001-81-M-A256, January 1982.

35. K. M. Chandy and J. Misra, Asynchronous Distributed Simulation via a Sequence of Parallel Computations, Communications of the ACM, Vol. 24, No. 11, April 1981.

36. M. Gerla and L. Kleinrock, On the Topological Design of Distributed Computer Networks, IEEE Transactions on Communications, Vol. COM-25, No. 1, January 1977.

37. Mischa Schwartz, Computer-Communication Network Design and Analysis, Prentice-Hall, New Jersey, 1977.

38. Jeanne K. Tamaki, Finite State Machines in Protocol Validation, NOSC Technical Report 842, 10 September 1982.

39. A. Danthine, Protocol Representation with Finite State Models, IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980.

40. Carl A. Sunshine, Survey of Protocol Definition and Verification Techniques, in Advances in Computer Communications and Networking, edited by Wesley Chu, Artech House, Denham, MA, 1977.

41. P. M. Merlin, Specification and Validation of Protocols, IEEE Transactions on Communications, Vol. COM-27, No. 11, November 1979.

42. E. Hayes, ACDS Data Requirements for a Performance Oriented Design (POD) Model, NOSC Memorandum of 20 January 1982.

43. POD Concept Paper, BQS Systems, Lincoln, MA, 31 August 1981.

44. GRINDER User's and Programmer's Guide, June 1977.

45. G. Bochman and C. Sunshine, Formal Methods in Communication Protocol Design, IEEE Transactions on Communications, Vol. COM-27, No. 4, April 1980.

46. R. R. Boorstyn and H. Frank, Large-Scale Network Topological Optimization, IEEE Transactions on Communications, Vol. COM-25, No. 1, January 1977.

47. Gerald Estrin, Modeling for Synthesis - The Gap Between Intent and Behavior, Proceedings of Symposium on Design Automation and Microprocessors, Palo Alto, CA, February 1977.

48. Howard Frand and Wushow Chou, Topological Optimization of Computer Networks, Proceedings of the IEEE, Vol. 60, No. 11, November 1972.

49. O. Spanio, Modeling of Local Computer Networks, Computer Networks, North-Holland Publishing Company, Vol. 3, 1979.

50. A. S. Tannenbaum, Computer Networks, Prentice-Hall, New Jersey, 1981.